**Лекция 3.1: Исключения / try-catch-finally. ‘**

**Синтаксис – это то что пропускает калькулятор**

**Семантика – как рекомендуються использовать эти конструкции**

[**0. Введение**](http://www.golovachcourses.com/exceptions-try-catch-finally/#0)

….[**0.1 Устойчивость vs корректность**](http://www.golovachcourses.com/exceptions-try-catch-finally/#0.1)

….[**0.2 Аналогия со стиральной машиной**](http://www.golovachcourses.com/exceptions-try-catch-finally/#0.2)

….**[0.3 Ключевые слова: try, catch, finally, throw, throws](http://www.golovachcourses.com/exceptions-try-catch-finally/" \l "0.3)**

….[**0.4 Почему используем System.err, а не System.out**](http://www.golovachcourses.com/exceptions-try-catch-finally/#0.4)

[**1. Компилятор требует вернуть результат (или требует молчать)**](http://www.golovachcourses.com/exceptions-try-catch-finally/#1)

[**2. Нелокальная передача управления (non-local control transfer)**](http://www.golovachcourses.com/exceptions-try-catch-finally/#2)

[**3. try + catch (catch – полиморфен)**](http://www.golovachcourses.com/exceptions-try-catch-finally/#3)

[**4. try + catch + catch + …**](http://www.golovachcourses.com/exceptions-try-catch-finally/#4)

[**5. try + finally**](http://www.golovachcourses.com/exceptions-try-catch-finally/#5)

[**6. try + catch + finally**](http://www.golovachcourses.com/exceptions-try-catch-finally/#6)

[**7. Вложенные try + catch + finally**](http://www.golovachcourses.com/exceptions-try-catch-finally/#7)

**0. Введение**

**0.1 Устойчивость vs корректность**

В программировании, как и во всякой инженерной дисциплине, есть много дилемм проектирования.  
Одна из них

УСТОЙЧИВОСТЬ или КОРРЕКТНОСТЬ

Система **устойчива**, если при любых входных данных она продолжает функционировать (возможно, в неверном режиме).  
Система **корректна**, если все результаты, которые она выдает – верные (но может “упасть” или “повиснуть” и не выдать результата вовсе).

Пример:  
Самонаводящаяся ракета класса воздух-воздух (т.е. выстрел производится из самолета по самолету) на 1 секунду упустила из виду цель и позже вновь ее захватила. С большой долей вероятности ракета все еще летит в противника, но, с другой стороны, возможно уже летит назад, в того, кто ее запустил.  
**Корректность**: так как мы не уверены на 100%, что летим в противника – прекратить преследование и самоликвидироваться.  
**Устойчивость**: так как мы не хотим, что бы наши ракеты самоликвидировались из-за различных случайностей (стрельба в облаках, отблески от водной поверхности, повышенная влажность, солнечная активность, . …) то, учитывая, что цель потеряли “всего-то” на 1 секунду и с большой долей вероятности (> 95%) все еще летим в противника, продолжаем преследование.

Пример:  
Кардиостимулятор обнаружил, что датчик пульса “выдает” 1.000.000 ударов сердца в секунду (хотя ожидается, что нормальный диапазон – 30-300 ударов в секунду). Алгоритм стимулирования сердечной деятельности верен только для диапазона 30-300 ударов в секунду. Программа кардиостимулятора “понимает”, что датчик неисправен и, как следствие, она не может “отработать четко по алгоритму”, т.е. быть корректной. Но, полагаю, от нее требуется быть устойчивой, т.е. продолжать функционировать в любых условиях исходя из наиболее вероятных предположений. Скажем выдавать сигнал так, как будто у больного 80 ударов сердца в секунду (хотя у больного реальный пульс – 75 ударов в секунду и значит мы не совсем корректно работаем).

**0.2 Аналогия со стиральной машиной**

При любом напряжении стиральная машина подает его на двигатель. Сгорит или нет – как фишка ляжет.

|  |  |
| --- | --- |
|  | **public class WashingMachine {**  **private Engine engine = new Engine();**  **public void wash(int voltage) {**  **// Бог даст, повезет**  **engine.run(voltage);**  **}**  **}** |

Если напряжение превышает лимит (voltage > 230), то помигает красной лампочкой (System.put.println) и но в любом случае подам его на двигатель. Может сгорит двигатель, а может и нет

|  |  |
| --- | --- |
|  | **public class WashingMachine {**  **private Engine engine = new Engine();**  **public void wash(int voltage) {**  **if (voltage > 230) { // Напряжение великовато**  **// тебе в консоль пишу я**  **System.out.println("Voltage over limit: " + voltage);**  **}**  **// но черт с ним, рискну я**  **engine.run(voltage);**  **}**  **}** |

Если напряжение превышает лимит (voltage > 230), то помигает красной лампочкой (System.put.println) и не будет подавать его на двигатель. Но если вы не следили за лампочкой, то можете утром найти грязное белье (engine не запускался)

|  |  |  |
| --- | --- | --- |
|  |  | **public class WashingMachine {**  **private Engine engine = new Engine();**  **public void wash(int voltage) {**  **if (voltage > 230) { // Напряжение великовато**  **// тебе в консоль пишу я**  **System.out.println("Voltage over limit: " + voltage);**  **} else {**  **// когда великовато напряжение - НЕЗАМЕТНО НЕ СТИРАЮ твое белье я**  **engine.run(voltage);**  **}**  **}**  **}** |

Если напряжение превышает лимит (voltage > 230), то вместо стирки – взорву твой дом (System.exit())

|  |  |
| --- | --- |
|  | **public class WashingMachine {**  **private Engine engine = new Engine();**  **public void wash(int voltage) {**  **if (voltage > 230) { // Напряжение великовато**  **// кабы чего не случилось - взорву твой дом я**  **System.exit(42);**  **}**  **engine.run(voltage);**  **}**  **}** |

Если напряжение превышает лимит (voltage > 230), то вместо стирки – разбросает белье по комнате, не проигнорируешь

|  |  |
| --- | --- |
|  | // unchecked - берегу твои нервы в COMPILE-**TIME**  **class VoltageException extends RuntimeException {}**    **public class WashingMachine {**  **private Engine engine = new Engine();**  **// нет throws - тут не бойся в COMPILE-TIME ты**  **public void wash(int voltage) {**  **if (voltage > 230) { // Напряжение великовато**  **// и крутиться не буду и не проигнорируешь меня в RUNTIME ты**  **throw new VoltageException("Voltage over limit: " + voltage);**  **}**  **engine.run(voltage);**  **}**  **}** |

Если напряжение превышает лимит (voltage > 230), то вместо стирки – разбросает белье по комнате, не проигнорируешь. А перед каждой стиркой (вызов метода wash()) будешь расписываться в ведомости, что ознакомлен с инструкции по эксплуатации (отреагировать на throws) и заешь, что в случае превышения напряжения – тебе разбросают белье по ванной

|  |  |
| --- | --- |
|  | // checked - буду пугать тебя в COMPILE-TIME  **class VoltageException extends Exception {}**    **public class WashingMachine {**  **private Engine engine = new Engine();**  **// throws - вот тут бойся в COMPILE-TIME ты**  **public void wash(int voltage) throws VoltageException {**  **if (voltage > 230) {**  **// и крутиться не буду и не проигнорируешь меня в RUNTIME ты**  **throw new VoltageException("Voltage over limit: " + voltage);**  **}**  **engine.run(voltage);**  **}**  **}** |

**0.3 Ключевые слова: try, catch, finally, throw, throws**

Механизм исключительных ситуаций в Java поддерживается пятью ключевыми словами

try, catch, finally, throw, throws

“Магия” (т.е. некоторое поведение никак не отраженное в исходном коде и потому неповторяемое пользователем) исключений #1 заключается в том, что catch, throw, throws можно использовать исключительно с java.lang.Throwable или его потомками.

**throws:**  
Годится

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) throws Throwable {}**  **}** |

Не годится?

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) throws String {}**  **}**    >> COMPILATION ERROR: Incompatible types: required 'java.lang.Throwable', found: 'java.lang.String' |

**catch:**  
Годится

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **} catch (Throwable t) {}**  **}**  **}** |

Не годится

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **} catch (String s) {}**  **}**  **}**    >> COMPILATION ERROR: Incompatible types: required 'java.lang.Throwable', found: 'java.lang.String' |

**throw:**  
Годится

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **// Error - потомок Throwable**  **throw new Error();**  **}**  **}** |

Не годится

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **throw new String("Hello!");**  **}**  **}**    >> COMPILATION ERROR: Incompatible types: required 'java.lang.Throwable', found: 'java.lang.String' |

Кроме того, throw требуется не-null аргумент, иначе NullPointerException в момент выполнения

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **throw null;**  **}**  **}**    >> RUNTIME ERROR: Exception in thread "main" java.lang.NullPointerException |

throw и new – это две независимых операции. В следующем коде мы независимо создаем объект исключения и “бросаем” его

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **Error ref = new Error(); // создаем экземпляр**  **throw ref;               // "бросаем" его**  **}**  **}**    >> RUNTIME ERROR: Exception in thread "main" java.lang.Error |

Однако, попробуйте проанализировать вот это

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **f(null);**  **}**  **public static void f(NullPointerException e) {**  **try {**  **throw e;**  **} catch (NullPointerException npe) {**  **f(npe);**  **}**  **}**  **}**    >> RUNTIME ERROR: Exception in thread "main" java.lang.StackOverflowError |

**0.4 Почему используем System.err, а не System.out**

System.out – buffered-поток вывода, а System.err – нет. Таким образом вывод может быть, как таким

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.out.println("sout");**  **throw new Error();**  **}**  **}**  >> RUNTIME ERROR: Exception in thread "main" java.lang.Error  >> sout |

Так и вот таким (err обогнало out при выводе в консоль)

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.out.println("sout");**  **throw new Error();**  **}**  **}**  >> sout  >> RUNTIME ERROR: Exception in thread "main" java.lang.Error |

Давайте это нарисуем

|  |  |
| --- | --- |
|  | буфер сообщений                      +----------------+                   +->| msg2 msg1 msg0 | --> out                  /   +----------------+        \                 /                                 +-> +--------+  ВАШЕ ПРИЛОЖЕНИЕ                                      | КОНСОЛЬ|                 \                                 +-> +--------+                  \                               /                   +------------------------> err                   нет буфера, сразу печатаем |

когда Вы пишете в System.err – ваше сообщение тут же выводится на консоль, но когда пишете в System.out, то оно может на какое-то время быть буферизированно. Stacktrace необработанного исключение выводится через System.err, что позволяет им обгонять “обычные” сообщения.

**1. Компилятор требует вернуть результат (или требует молчать)**

Если в объявлении метода сказано, что он возвращает НЕ void, то компилятор зорко следит, что бы мы вернули экземпляр требуемого типа или экземпляр типа, который можно неявно привести к требуемому

|  |  |
| --- | --- |
|  | **public class App {**  **public double sqr(double arg) { // надо double**  **return arg \* arg;           // double \* double - это double**  **}**  **}** |

|  |  |
| --- | --- |
|  | **public class App {**  **public double sqr(double arg) { // надо double**  **int k = 1;                  // есть int**  **return k;                   // можно неявно преобразовать int в double**  **}**  **}** |

|  |  |
| --- | --- |
|  | // на самом деле, компилятор сгенерирует байт-код для следующих исходников (???)  **public class App {**  **public double sqr(double arg) { // надо double**  **int k = 1;                  // есть int**  **return (double) k;          // явное преобразование int в double**  **}**  **}** |

вот так не пройдет (другой тип)

|  |  |
| --- | --- |
|  | **public class App {**  **public static double sqr(double arg) {**  **return "hello!";**  **}**  **}**    >> COMPILATION ERROR: Incompatible types. Required: double. Found: java.lang.String |

Вот так не выйдет – нет возврата

|  |  |
| --- | --- |
|  | public class App {      public static double sqr(double arg) {      }  }    >> COMPILATION ERROR: Missing return statement |

и вот так не пройдет (компилятор не может удостовериться, что возврат будет)

|  |  |
| --- | --- |
|  | **public class App {**  **public static double sqr(double arg) {**  **if (System.currentTimeMillis() % 2 == 0) {**  **return arg \* arg; // если currentTimeMillis() - четное число, то все ОК**  **}**  **// а если нечетное, что нам возвращать?**  **}**  **}**    >> COMPILATION ERROR: Missing return statement |

Компилятор отслеживает, что бы мы что-то вернули, так как иначе непонятно, что должна была бы напечатать данная программа

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **double d = sqr(10.0); // ну, и чему равно d?**  **System.out.println(d);**  **}**  **public static double sqr(double arg) {**  **// nothing**  **}**  **}**    >> COMPILATION ERROR: Missing return statement |

Из-забавного, можно ничего не возвращать, а “повесить метод”

|  |  |
| --- | --- |
|  | **public class App {**  **public static double sqr(double arg) {**  **while (true); // Удивительно, но КОМПИЛИРУЕТСЯ!**  **}**  **}** |

Тут в d никогда ничего не будет присвоено, так как метод sqr повисает

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **double d = sqr(10.0);  // sqr - навсегда "повиснет", и**  **System.out.println(d); // d - НИКОГДА НИЧЕГО НЕ БУДЕТ ПРИСВОЕНО!**  **}**  **public static double sqr(double arg) {**  **while (true); // Вот тут мы на века "повисли"**  **}**  **}** |

Компилятор пропустит “вилку” (таки берем в квадрат ИЛИ висим)

|  |  |
| --- | --- |
|  | **public class App {**  **public static double sqr(double arg) {**  **if (System.currentTimeMillis() % 2 == 0) {**  **return arg \* arg; // ну ладно, вот твой double**  **} else {**  **while (true);     // а тут "виснем" навсегда**  **}**  **}**  **}** |

**Но механизм исключений позволяет НИЧЕГО НЕ ВОЗВРАЩАТЬ!**

|  |
| --- |
| **public class App {**  **public static double sqr(double arg) {**  **throw new RuntimeException();**  **}**  **}** |

Итак, у нас есть ТРИ варианта для компилятора

|  |  |
| --- | --- |
|  | **public class App {**  **public static double sqr(double arg) {// согласно объявлению метода ты должен вернуть double**  **long time = System.currentTimeMillis();**  **if (time % 2 == 0) {**  **return arg \* arg;             // ок, вот твой double**  **} else if (time % 2 == 1) { {**  **while (true);                 // не, я решил "повиснуть"**  **} else {**  **throw new RuntimeException(); // или бросить исключение**  **}**  **}**  **}** |

**Но КАКОЙ ЖЕ double вернет функция, бросающая RuntimeException?  
А НИКАКОЙ!**

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **// sqr - "сломается" (из него "выскочит" исключение),**  **double d = sqr(10.0);  // выполнение метода main() прервется в этой строчке и**  **// d - НИКОГДА НИЧЕГО НЕ БУДЕТ ПРИСВОЕНО!**  **System.out.println(d); // и печатать нам ничего не придется!**  **}**  **public static double sqr(double arg) {**  **throw new RuntimeException(); // "бросаем" исключение**  **}**  **}**    >> RUNTIME ERROR: Exception in thread "main" java.lang.RuntimeException |

Подытожим: бросаемое исключение – это дополнительный возвращаемый тип. Если ваш метод объявил, что возвращает double, но у вас нет double – можете бросить исключение. Если ваш метод объявил, что ничего не возвращает (void), но у вам таки есть что сказать – можете бросить исключение.

Давайте рассмотрим некоторый пример из практики.

Задача: реализовать функцию, вычисляющую площадь прямоугольника

|  |  |
| --- | --- |
|  | **public static int area(int width, int height) {...}** |

важно, что задание звучит именно так, в терминах предметной области – “вычислить площадь прямоугольника”, а не в терминах решения “перемножить два числа”:

|  |
| --- |
| **public static int area(int width, int height) {**  **return width \* height; // тут просто перемножаем**  **}** |

Вопрос: что делать, если мы обнаружили, что хотя бы один из аргументов – отрицательное число?  
Если просто умножить, то мы пропустили ошибочные данные дальше. Что еще хуже, возможно, мы “исправили ситуацию” – сказали что площадь прямоугольника с двумя отрицательными сторонами -10 и -20 = 200.

Мы не можем ничего не вернуть

|  |  |
| --- | --- |
|  | public static int area(int width, int height) {      if (width < 0 || height < 0) {          // у вас плохие аргументы, извините      } else {          return width \* height;      }  }    >> COMPILATION ERROR: Missing return statement |

Можно, конечно, отписаться в консоль, но кто ее будет читать и как определить где была поломка. При чем, вычисление то продолжится с неправильными данными

|  |  |
| --- | --- |
|  | public static int area(int width, int height) {      if (width < 0 || height < 0) {          System.out.println("Bad ...");      }      return width \* height;  } |

Можно вернуть специальное значение, показывающее, что что-то не так (error code), но кто гарантирует, что его прочитают, а не просто воспользуются им?

|  |  |
| --- | --- |
|  | **public static int area(int width, int height) {**  **if (width < 0 || height < 0) {**  **return -1; // специальное "неправильное" значение площади**  **}**  **return width \* height;**  **}** |

Можем, конечно, целиком остановить виртуальную машину

|  |  |
| --- | --- |
|  | **public static int area(int width, int height) {**  **if (width < 0 || height < 0) {**  **System.exit(0);**  **}**  **return width \* height;**  **}** |

Но "правильный путь" таков: если обнаружили возможное некорректное поведение, то  
1. Вычисления остановить, сгенерировать сообщение-поломку, которое трудно игнорировать, предоставить пользователю информацию о причине, предоставить пользователю возможность все починить (загрузить белье назад и повторно нажать кнопку старт)

|  |  |
| --- | --- |
|  | **public static int area(int width, int height) {**  **if (width < 0 || height < 0) {**  **throw new IllegalArgumentException("Negative sizes: w = " + width + ", h = " + height);**  **}**  **return width \* height;**  **}** |

**2. Нелокальная передача управления (non-local control transfer)**

**Механизм исключительных ситуация (исключений) - это механизм НЕЛОКАЛЬНОЙ ПЕРЕДАЧИ УПРАВЛЕНИЯ.**  
Что под этим имеется в виду?  
Программа, в ходе своего выполнения (точнее исполнения инструкций в рамках отдельного потока), оперирует стеком ("стопкой") фреймов. Передача управления осуществляется либо в рамках одного фрейма

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **// Пример: ОПЕРАТОР ПОСЛЕДОВАТЕЛЬНОСТИ**  **int x = 42;    // первый шаг**  **int y = x \* x; // второй шаг**  **x = x \* y;     // третий шаг**  **...**  **}**  **}** |

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **// Пример: ОПЕРАТОР ВЕТВЛЕНИЯ**  **if (args.length > 2) { первый шаг**  **// второй шаг или тут**  **...**  **} else {**  **// или тут**  **...**  **}**  **// третий шаг**  **...**  **}**  **}** |

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **// Пример: ОПЕРАТОР ЦИКЛА do..while**  **int x = 1;**  **do {**  **...**  **} while (x++ < 10);**  **...**  **}**  **}** |

и другие операторы.

Либо передача управления происходит в "стопке" фреймов между СОСЕДНИМИ фреймами  
....1) вызов метода: создаем новый фрейм, помещаем его на верхушку стека и переходим в него  
....2) выход из метода: возвращаемся к предыдущему фрейму (через return или просто кончились инструкции в методе)

return - выходим из ОДНОГО фрейма (из фрейма #4(метод h()))

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.err.println("#1.in");**  **f(); // создаем фрейм, помещаем в стек, передаем в него управление**  **System.err.println("#1.out"); // вернулись**  **} // выходим из текущего фрейма, кончились инструкции**    **public static void f() {**  **System.err.println(".   #2.in");**  **g(); // создаем фрейм, помещаем в стек, передаем в него управление**  **System.err.println(".   #2.out");  //вернулись**  **} // выходим из текущего фрейма, кончились инструкции**    **public static void g() {**  **System.err.println(".   .   #3.in");**  **h(); // создаем фрейм, помещаем в стек, передаем в него управление**  **System.err.println(".   .   #3.out"); // вернулись**  **} // выходим из текущего фрейма, кончились инструкции**    **public static void h() {**  **System.err.println(".   .   .   #4.in");**  **if (true) {**  **System.err.println(".   .   .   #4.RETURN");**  **return; // выходим из текущего фрейма по 'return'**  **}**  **System.err.println(".   .   .   #4.out"); // ПРОПУСКАЕМ**  **}**  **}**    **>> #1.in**  **>> .   #2.in**  **>> .   .   #3.in**  **>> .   .   .   #4.in**  **>> .   .   .   #4.RETURN**  **>> .   .   #3.out**  **>> .   #2.out**  **>> #1.out** |

throw - выходим из ВСЕХ фреймов

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.err.println("#1.in");**  **f(); // создаем фрейм, помещаем в стек, передаем в него управление**  **System.err.println("#1.out"); // ПРОПУСТИЛИ!**  **}**    **public static void f() {**  **System.err.println(".   #2.in");**  **g(); // создаем фрейм, помещаем в стек, передаем в него управление**  **System.err.println(".   #2.out"); // ПРОПУСТИЛИ!**  **}**    **public static void g() {**  **System.err.println(".   .   #3.in");**  **h(); // создаем фрейм, помещаем в стек, передаем в него управление**  **System.err.println(".   .   #3.out"); // ПРОПУСТИЛИ!**  **}**    **public static void h() {**  **System.err.println(".   .   .   #4.in");**  **if (true) {**  **System.err.println(".   .   .   #4.THROW");**  **throw new Error(); // выходим со всей пачки фреймов ("раскрутка стека") по 'throw'**  **}**  **System.err.println(".   .   .   #4.out"); // ПРОПУСТИЛИ!**  **}**  **}**    >> #1.in  >> .   #2.in  >> .   .   #3.in  >> .   .   .   #4.in  >> .   .   .   #4.THROW  >> RUNTIME ERROR: Exception in thread "main" java.lang.Error |

При помощи catch мы можем остановить летящее исключение (причина, по которой мы автоматически покидаем фреймы).  
Останавливаем через 3 фрейма, пролетаем фрейм #4(метод h()) + пролетаем фрейм #3(метод g()) + фрейм #2(метод f())

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.err.println("#1.in");**  **try {**  **f(); // создаем фрейм, помещаем в стек, передаем в него управление**  **} catch (Error e) { // "перехватили" "летящее" исключение**  **System.err.println("#1.CATCH");  // и работаем**  **}**  **System.err.println("#1.out");  // работаем дальше**  **}**    **public static void f() {**  **System.err.println(".   #2.in");**  **g(); // создаем фрейм, помещаем в стек, передаем в него управление**  **System.err.println(".   #2.out"); // ПРОПУСТИЛИ!**  **}**    **public static void g() {**  **System.err.println(".   .   #3.in");**  **h(); // создаем фрейм, помещаем в стек, передаем в него управление**  **System.err.println(".   .   #3.out"); // ПРОПУСТИЛИ!**  **}**    **public static void h() {**  **System.err.println(".   .   .   #4.in");**  **if (true) {**  **System.err.println(".   .   .   #4.THROW");**  **throw new Error(); // выходим со всей пачки фреймов ("раскрутка стека") по 'throw'**  **}**  **System.err.println(".   .   .   #4.out"); // ПРОПУСТИЛИ!**  **}**  **}**    >> #1.in  >> .   #2.in  >> .   .   #3.in  >> .   .   .   #4.in  >> .   .   .   #4.THROW  >> #1.CATCH  >> #1.out |

Обратите внимание, стандартный сценарий работы был восстановлен в методе main() (фрейм #1)

Останавливаем через 2 фрейма, пролетаем фрейм #4(метод h()) + пролетаем фрейм #3(метод g())

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.err.println("#1.in");**  **f(); // создаем фрейм, помещаем в стек, передаем в него управление**  **System.err.println("#1.out"); // вернулись и работаем**  **}**    **public static void f() {**  **System.err.println(".   #2.in");**  **try {**  **g(); // создаем фрейм, помещаем в стек, передаем в него управление**  **} catch (Error e) { // "перехватили" "летящее" исключение**  **System.err.println(".   #2.CATCH");  // и работаем**  **}**  **System.err.println(".   #2.out");  // работаем дальше**  **}**    **public static void g() {**  **System.err.println(".   .   #3.in");**  **h(); // создаем фрейм, помещаем в стек, передаем в него управление**  **System.err.println(".   .   #3.out"); // ПРОПУСТИЛИ!**  **}**    **public static void h() {**  **System.err.println(".   .   .   #4.in");**  **if (true) {**  **System.err.println(".   .   .   #4.THROW");**  **throw new Error(); // выходим со всей пачки фреймов ("раскрутка стека") по 'throw'**  **}**  **System.err.println(".   .   .   #4.out"); // ПРОПУСТИЛИ!**  **}**  **}**    >> #1.in  >> .   #2.in  >> .   .   #3.in  >> .   .   .   #4.in  >> .   .   .   #4.THROW  >> .   #2.CATCH  >> .   #2.out  >> #1.out |

Останавливаем через 1 фрейм (фактически аналог return, просто покинули фрейм "другим образом")

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.err.println("#1.in");**  **f(); // создаем фрейм, помещаем в стек, передаем в него управление**  **System.err.println("#1.out"); // вернулись и работаем**  **}**    **public static void f() {**  **System.err.println(".   #2.in");**  **g(); // создаем фрейм, помещаем в стек, передаем в него управление**  **System.err.println(".   #2.out"); // вернулись и работаем**  **}**    **public static void g() {**  **System.err.println(".   .   #3.in");**  **try {**  **h(); // создаем фрейм, помещаем в стек, передаем в него управление**  **} catch (Error e) { // "перехватили" "летящее" исключение**  **System.err.println(".   .   #3.CATCH");  // и работаем**  **}**  **System.err.println(".   .   #3.out");  // работаем дальше**  **}**    **public static void h() {**  **System.err.println(".   .   .   #4.in");**  **if (true) {**  **System.err.println(".   .   .   #4.THROW");**  **throw new Error(); // выходим со всей пачки фреймов ("раскрутка стека") по 'throw'**  **}**  **System.err.println(".   .   .   #4.out"); // ПРОПУСТИЛИ!**  **}**  **}**    >> #1.in  >> .   #2.in  >> .   .   #3.in  >> .   .   .   #4.in  >> .   .   .   #4.THROW  >> .   .   #3.CATCH  >> .   .   #3.out  >> .   #2.out  >> #1.out |

Итак, давайте сведем все на одну картинку

|  |  |
| --- | --- |
| 2 | // ---Используем RETURN--- // ---Используем THROW---  // Выход из 1-го фрейма    // Выход из ВСЕХ (из 4) фреймов  #1.in                        #1.in  .   #2.in                    .   #2.in  .   .   #3.in                .   .   #3.in  .   .   .   #4.in            .   .   .   #4.in  .   .   .   #4.RETURN        .   .   .   #4.THROW  .   .   #3.out               RUNTIME EXCEPTION: Exception in thread "main" java.lang.Error  .   #2.out  #1.out    // ---Используем THROW+CATCH---  // Выход из 3-х фреймов      // Выход из 2-х фреймов      // Выход из 1-го фрейма  #1.in                        #1.in                        #1.in  .   #2.in                    .   #2.in                    .   #2.in  .   .   #3.in                .   .   #3.in                .   .   #3.in  .   .   .   #4.in            .   .   .   #4.in            .   .   .   #4.in  .   .   .   #4.THROW         .   .   .   #4.THROW         .   .   .   #4.THROW  #1.CATCH                     .   #2.CATCH                 .   .   #3.CATCH  #1.out                       .   #2.out                   .   .   #3.out                               #1.out                       . #2.out                                                            #1.out |

**3. try + catch (catch - полиморфен)**

Напомним иерархию исключений

|  |  |
| --- | --- |
|  | Object          |      Throwable      /      \  Error     Exception                |        RuntimeException |

То, что исключения являются объектами важно для нас в двух моментах  
1. Они образуют иерархию с корнем java.lang.Throwable (java.lang.Object - предок java.lang.Throwable, но Object - уже не исключение)  
2. Они могут иметь поля и методы (этот вопрос [изучим позже](http://www.study.golovachcourses.com/exceptions-internals-and-handling/))

По первому пункту: catch - полиморфная конструкция, т.е. catch по типу Parent перехватывает летящие экземпляры любого типа, который является Parent-ом (т.е. экземпляры непосредственно Parent-а или любого потомка Parent-а)

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.print(" 0");**  **if (true) {throw new RuntimeException();}**  **System.err.print(" 1");**  **// catch по Exception перехватывает И RuntimeException**  **} catch (Exception e) {**  **System.err.print(" 2"); // сюда заходим**  **}**  **System.err.println(" 3");**  **}**  **}**    >> 0 2 3 |

**Даже так: в блоке catch мы будем иметь ссылку типа Exception на объект типа RuntimeException**

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **throw new RuntimeException();**  **} catch (Exception e) {**  **if (e instanceof RuntimeException) {**  **RuntimeException re = (RuntimeException) e;**  **System.err.print("Это RuntimeException на самом деле!!!");**  **} else {**  **System.err.print("В каком смысле не RuntimeException???");**  **}**  **}**  **}**  **}**    >> Это RuntimeException на самом деле!!! |

catch по потомку не может поймать предка

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) throws Exception { // пока игнорируйте 'throws'**  **try {**  **System.err.print(" 0");**  **if (true) {throw new Exception();}**  **System.err.print(" 1");**  **} catch (RuntimeException e) { // catch по Exception НЕ перехватывает RuntimeException**  **System.err.print(" 2");    // сюда НЕ заходим**  **}**  **System.err.print(" 3");**  **}**  **}**    >> 0  >> RUNTIME EXCEPTION: Exception in thread "main" java.lang.Exception |

catch по одному "брату" не может поймать другого "брата" (Error и Exception не находятся в отношении предок-потомок, они из параллельных веток наследования от Throwable)

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.print(" 0");**  **if (true) {throw new Error();}**  **System.err.print(" 1");**  **} catch (Exception e) {     // catch по Exception НЕ перехватывает Error**  **System.err.print(" 2"); // сюда НЕ заходим**  **}**  **System.err.print(" 3");**  **}**  **}**    >> 0  >> RUNTIME EXCEPTION: Exception in thread "main" java.lang.Error |

По предыдущим примерам - надеюсь вы обратили внимание, что если исключение перехвачено, то JVM выполняет операторы идущие ПОСЛЕ последних скобок try+catch.  
Но если не перехвачено, то мы  
1. не заходим в блок catch  
2. покидаем фрейм метода с летящим исключением

А что будет, если мы зашли в catch, и потом бросили исключение ИЗ catch?

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.print(" 0");**  **if (true) {throw new RuntimeException();}**  **System.err.print(" 1");**  **} catch (RuntimeException e) {     // перехватили RuntimeException**  **System.err.print(" 2");**  **if (true) {throw new Error();} // но бросили Error**  **}**  **System.err.println(" 3");          // пропускаем - уже летит Error**  **}**  **}**    >> 0 2  >> RUNTIME EXCEPTION: Exception in thread "main" java.lang.Error |

В таком случае выполнение метода тоже прерывается (не печатаем "3"). Новое исключение не имеет никакого отношения к try-catch

Мы можем даже кинуть тот объект, что у нас есть "на руках"

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.print(" 0");**  **if (true) {throw new RuntimeException();}**  **System.err.print(" 1");**  **} catch (RuntimeException e) { // перехватили RuntimeException**  **System.err.print(" 2");**  **if (true) {throw e;}       // и бросили ВТОРОЙ раз ЕГО ЖЕ**  **}**  **System.err.println(" 3");      // пропускаем - опять летит RuntimeException**  **}**  **}**    >> 0 2  >> RUNTIME EXCEPTION: Exception in thread "main" java.lang.RuntimeException |

И мы не попадем в другие секции catch, если они есть

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.print(" 0");**  **if (true) {throw new RuntimeException();}**  **System.err.print(" 1");**  **} catch (RuntimeException e) {     // перехватили RuntimeException**  **System.err.print(" 2");**  **if (true) {throw new Error();} // и бросили новый Error**  **} catch (Error e) { // хотя есть cath по Error "ниже", но мы в него не попадаем**  **System.err.print(" 3");**  **}**  **System.err.println(" 4");**  **}**  **}**    >> 0 2  >> RUNTIME EXCEPTION: Exception in thread "main" java.lang.Error |

Обратите внимание, мы не напечатали "3", хотя у нас летит Error а "ниже" расположен catch по Error. Важный момент заключается в том, что catch имеет отношение исключительно к try-секции, но не к другим catch-секциям.

Как покажем ниже - можно строить вложенные конструкции, но вот пример, "исправляющий" эту ситуацию

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.print(" 0");**  **if (true) {throw new RuntimeException();}**  **System.err.print(" 1");**  **} catch (RuntimeException e) { // перехватили RuntimeException**  **System.err.print(" 2.1");**  **try {**  **System.err.print(" 2.2");**  **if (true) {throw new Error();} // и бросили новый Error**  **System.err.print(" 2.3");**  **} catch (Throwable t) {            // перехватили Error**  **System.err.print(" 2.4");**  **}**  **System.err.print(" 2.5");**  **} catch (Error e) { // хотя есть cath по Error "ниже", но мы в него не попадаем**  **System.err.print(" 3");**  **}**  **System.err.println(" 4");**  **}**  **}**    >> 0 2.1 2.2 2.4 2.5 4 |

**Тест**

Для прохождения теста по теме Ex.TryCatch (уровень сложности теста: Basic ) нажмите “Start Quiz”

**4. try + catch + catch + ...**

Как вы видели, мы можем расположить несколько catch после одного try.

Но есть такое правило - **нельзя ставить потомка после предка!** (RuntimeException после Exception)

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **} catch (Exception e) {**  **} catch (RuntimeException e) {**  **}**  **}**  **}**    >> COMPILATION ERROR: Exception 'java.lang.RuntimeException' has alredy been caught |

Ставить брата после брата - можно (RuntimeException и Error - "братья", они из параллельных веток наследования)

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **} catch (Error e) {**  **} catch (RuntimeException e) {**  **}**  **}**  **}** |

Как происходит выбор "правильного" catch? Да очень просто - JVM идет сверху-вниз до тех пор, пока не найдет такой catch что в нем указано ваше исключение или его предок - туда и заходит. Ниже - не идет.

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **throw new Exception();**  **} catch (RuntimeException e) {**  **System.err.println("catch RuntimeException");**  **} catch (Exception e) {**  **System.err.println("catch Exception");**  **} catch (Throwable e) {**  **System.err.println("catch Throwable");**  **}**  **System.err.println("next statement");**  **}**  **}**    >> catch Exception  >> next statement |

Выбор catch осуществляется в runtime (а не в compile-time), значит учитывается не тип ССЫЛКИ (Throwable), а тип ССЫЛАЕМОГО (Exception)

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **Throwable t = new Exception(); // ссылка типа Throwable на объект типа Exception**  **throw t;**  **} catch (RuntimeException e) {**  **System.err.println("catch RuntimeException");**  **} catch (Exception e) { // попадем сюда**  **System.err.println("catch Exception");**  **} catch (Throwable e) { // а не сюда**  **System.err.println("catch Throwable");**  **}**  **System.err.println("next statement");**  **}**  **}**    >> catch Exception  >> next statement |

**5. try + finally**

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

finally-секция получает управление, если try-блок завершился успешно

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.println("try");**  **} finally {**  **System.err.println("finally");**  **}**  **}**  **}**    >> try  >> finally |

finally-секция получает управление, даже если try-блок завершился исключением

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **throw new RuntimeException();**  **} finally {**  **System.err.println("finally");**  **}**  **}**  **}**   >> finally  >> Exception in thread "main" java.lang.RuntimeException |

finally-секция получает управление, даже если try-блок завершился директивой выхода из метода

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **return;**  **} finally {**  **System.err.println("finally");**  **}**  **}**  **}**    >> finally |

finally-секция НЕ вызывается только если мы "прибили" JVM

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.exit(42);**  **} finally {**  **System.err.println("finally");**  **}**  **}**  **}**    >> Process finished with exit code 42 |

System.exit(42) и Runtime.getRuntime().exit(42) - это синонимы

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **Runtime.getRuntime().exit(42);**  **} finally {**  **System.err.println("finally");**  **}**  **}**  **}**    >> Process finished with exit code 42 |

И при Runtime.getRuntime().halt(42) - тоже не успевает зайти в finally

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **Runtime.getRuntime().halt(42);**  **} finally {**  **System.err.println("finally");**  **}**  **}**  **}**    >> Process finished with exit code 42 |

**exit() vs halt()**  
javadoc: java.lang.Runtime#halt(int status)  
... Unlike the exit method, this method does not cause shutdown hooks to be started and does not run uninvoked finalizers if finalization-on-exit has been enabled. If the shutdown sequence has already been initiated then this method does not wait for any running shutdown hooks or finalizers to finish their work.  
??? - объяснить, что это значит

Однако finally-секция не может "починить" try-блок завершившийся исключением (заметьте, "more" - не выводится в консоль)

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.println("try");**  **if (true) {throw new RuntimeException();}**  **} finally {**  **System.err.println("finally");**  **}**  **System.err.println("more");**  **}**  **}**    >> try  >> finally  >> Exception in thread "main" java.lang.RuntimeException |

Трюк с "if (true) {...}" требуется, так как иначе компилятор обнаруживает недостижимый код (последняя строка) и отказывается его компилировать

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.println("try");**  **throw new RuntimeException();**  **} finally {**  **System.err.println("finally");**  **}**  **System.err.println("more"); // тут ошибка компиляции**  **}**  **}**    >> COMPILER ERROR: Unrechable statement |

И finally-секция не может "предотвратить" выход из метода, если try-блок вызвал return ("more" - не выводится в консоль)

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.println("try");**  **if (true) {return;}**  **} finally {**  **System.err.println("finally");**  **}**  **System.err.println("more");**  **}**  **}**    >> try  >> finally |

Однако finally-секция может "перебить" throw/return при помощи другого throw/return

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.err.println(f());**  **}**  **public static int f() {**  **try {**  **return 0;**  **} finally {**  **return 1;**  **}**  **}**  **}**    >> 1 |

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.err.println(f());**  **}**  **public static int f() {**  **try {**  **throw new RuntimeException();**  **} finally {**  **return 1;**  **}**  **}**  **}**    >> 1 |

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.err.println(f());**  **}**  **public static int f() {**  **try {**  **return 0;**  **} finally {**  **throw new RuntimeException();**  **}**  **}**  **}**    >> Exception in thread "main" java.lang.RuntimeException |

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.err.println(f());**  **}**  **public static int f() {**  **try {**  **throw new Error();            // ты был в начале**  **} finally {**  **throw new RuntimeException(); // но ты его "перебил"**  **}**  **}**  **}**   >> Exception in thread "main" java.lang.RuntimeException |

finally-секция может быть использована для завершающего действия, которое гарантированно будет вызвано (даже если было брошено исключение или автор использовал return) по окончании работы

|  |  |
| --- | --- |
|  | // open some resource  try {      // use resource  } finally {      // close resource  } |

Например, для освобождения захваченной блокировки

|  |
| --- |
| **Lock lock = new ReentrantLock();**  **...**  **lock.lock();**  **try {**  **// some code**  **} finally {**  **lock.unlock();**  **}** |

Или для закрытия открытого файлового потока

|  |  |
| --- | --- |
|  | **InputStream input = new FileInputStream("...");**  **try {**  **// some code**  **} finally {**  **input.close();**  **}** |

Специально для этих целей в Java 7 появилась конструкция [try-with-resources](http://www.golovachcourses.com/try-with-resources), ее мы изучим позже.

Вообще говоря, в finally-секция нельзя стандартно узнать было ли исключение.  
Конечно, можно постараться написать свой "велосипед".

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **System.err.println(f());**  **}**  **public static int f() {**  **long rnd = System.currenttimeMillis();**  **boolean finished = false;**  **try {**  **if (rnd % 3 == 0) {**  **throw new Error();**  **} else if (rnd % 3 == 1) {**  **throw new RuntimeException();**  **} else {**  **// nothing**  **}**  **finished = true;**  **} finally {**  **if (finished) {**  **// не было исключений**  **} else {**  **// было исключение, но какое?**  **}**  **}**  **}**  **}** |

**Нерекомендуемые практики**  
- return из finally-секции (можем затереть исключение из try-блока)  
- действия в finally-секции, которые могут бросить исключение (можем затереть исключение из try-блока)

**Тест**

Для прохождения теста по теме Ex.TryFinally (уровень сложности теста: Basic) нажмите “Start Quiz”

**6. try + catch + ... + catch + finally**

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

Нет исключения

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.print(" 0");**  **// nothing**  **System.err.print(" 1");**  **} catch(Error e) {**  **System.err.print(" 2");**  **} finally {**  **System.err.print(" 3");**  **}**  **System.err.print(" 4");**  **}**  **}**    **>> 0 1 3 4** |

Не заходим в catch, заходим в finally, продолжаем после оператора

Есть исключение и есть подходящий catch

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          try {              System.err.print(" 0");              if (true) {throw new Error();}              System.err.print(" 1");          } catch(Error e) {              System.err.print(" 2");          } finally {              System.err.print(" 3");          }          System.err.print(" 4");      }  }    >> 0 2 3 4 |

Заходим в catch, заходим в finally, продолжаем после оператора

Есть исключение но нет подходящего catch

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          try {              System.err.print(" 0");              if (true) {throw new RuntimeException();}              System.err.print(" 1");          } catch(Error e) {              System.err.print(" 2");          } finally {              System.err.print(" 3");          }          System.err.print(" 4");      }  }    >> 0 3  >> RUNTIME ERROR: Exception in thread "main" java.lang.RuntimeException |

Не заходим в catch, заходим в finally, не продолжаем после оператора - вылетаем с неперехваченным исключением

**Тест**

Для прохождения теста по теме Ex.TryCatchFinally.ControlFlow (уровень сложности теста: Mid) нажмите “Start Quiz”

**7. Вложенные try + catch + finally**

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

Операторы обычно допускают неограниченное вложение.  
Пример с if

|  |  |
| --- | --- |
|  | public class App {      public static void main(String[] args) {          if (args.length > 1) {              if (args.length > 2) {                  if (args.length > 3) {                      ...                  }              }          }      }  } |

**Пример с for**

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **for (int i = 0; i < 10; i++) {**  **for (int j = 0; j < 10; i++) {**  **for (int k = 0; k < 10; k++){**  **...**  **}**  **}**  **}**  **}**  **}** |

**Суть в том, что try-cacth-finally тоже допускает неограниченное вложение.  
Например вот так**

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **try {**  **try {**  **...**  **} catch (Exception e) {**  **} finally {}**  **} catch (Exception e) {**  **} finally {}**  **} catch (Exception e) {**  **} finally {}**  **}**  **}** |

**Или даже вот так**

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **try {**  **...**  **} catch (Exception e) {**  **...**  **} finally {**  **...**  **}**  **} catch (Exception e) {**  **try {**  **...**  **} catch (Exception e) {**  **...**  **} finally {**  **...**  **}**  **} finally {**  **try {**  **...**  **} catch (Exception e) {**  **...**  **} finally {**  **...**  **}**  **}**  **}**  **}** |

Ну что же, давайте исследуем как это работает.

Вложенный try-catch-finally без исключения

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.print(" 0");**  **try {**  **System.err.print(" 1");**  **// НИЧЕГО**  **System.err.print(" 2");**  **} catch (RuntimeException e) {**  **System.err.print(" 3"); // НЕ заходим - нет исключения**  **} finally {**  **System.err.print(" 4"); // заходим всегда**  **}**  **System.err.print(" 5");     // заходим - выполнение в норме**  **} catch (Exception e) {**  **System.err.print(" 6");     // НЕ заходим - нет исключения**  **} finally {**  **System.err.print(" 7");     // заходим всегда**  **}**  **System.err.print(" 8");         // заходим - выполнение в норме**  **}**  **}**    >> 0 1 2 4 5 7 8 |

Мы НЕ заходим в обе catch-секции (нет исключения), заходим в обе finally-секции и выполняем обе строки ПОСЛЕ finally.

Вложенный try-catch-finally с исключением, которое ПЕРЕХВАТИТ ВНУТРЕННИЙ catch

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.print(" 0");**  **try {**  **System.err.print(" 1");**  **if (true) {throw new RuntimeException();}**  **System.err.print(" 2");**  **} catch (RuntimeException e) {**  **System.err.print(" 3"); // ЗАХОДИМ - есть исключение**  **} finally {**  **System.err.print(" 4"); // заходим всегда**  **}**  **System.err.print(" 5");     // заходим - выполнение УЖЕ в норме**  **} catch (Exception e) {**  **System.err.print(" 6");     // не заходим - нет исключения, УЖЕ перехвачено**  **} finally {**  **System.err.print(" 7");     // заходим всегда**  **}**  **System.err.print(" 8");         // заходим - выполнение УЖЕ в норме**  **}**  **}**    >> 0 1 3 4 5 7 8 |

Мы заходим в ПЕРВУЮ catch-секцию (печатаем "3"), но НЕ заходим во ВТОРУЮ catch-секцию (НЕ печатаем "6", так как исключение УЖЕ перехвачено первым catch), заходим в обе finally-секции (печатаем "4" и "7"), в обоих случаях выполняем код после finally (печатаем "5"и "8", так как исключение остановлено еще первым catch).

Вложенный try-catch-finally с исключением, которое ПЕРЕХВАТИТ ВНЕШНИЙ catch

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.print(" 0");**  **try {**  **System.err.print(" 1");**  **if (true) {throw new Exception();}**  **System.err.print(" 2");**  **} catch (RuntimeException e) {**  **System.err.print(" 3"); // НЕ заходим - есть исключение, но НЕПОДХОДЯЩЕГО ТИПА**  **} finally {**  **System.err.print(" 4"); // заходим всегда**  **}**  **System.err.print(" 5"); // не заходим - выполнение НЕ в норме**  **} catch (Exception e) {**  **System.err.print(" 6"); // ЗАХОДИМ - есть подходящее исключение**  **} finally {**  **System.err.print(" 7"); // заходим всегда**  **}**  **System.err.print(" 8");  // заходим - выполнение УЖЕ в норме**  **}**  **}**    >> 0 1 4 6 7 8 |

Мы НЕ заходим в ПЕРВУЮ catch-секцию (не печатаем "3"), но заходим в ВТОРУЮ catch-секцию (печатаем "6"), заходим в обе finally-секции (печатаем "4" и "7"), в ПЕРВОМ случае НЕ выполняем код ПОСЛЕ finally (не печатаем "5", так как исключение НЕ остановлено), во ВТОРОМ случае выполняем код после finally (печатаем "8", так как исключение остановлено).

Вложенный try-catch-finally с исключением, которое НИКТО НЕ ПЕРЕХВАТИТ

|  |  |
| --- | --- |
|  | **public class App {**  **public static void main(String[] args) {**  **try {**  **System.err.print(" 0");**  **try {**  **System.err.print(" 1");**  **if (true) {throw new Error();}**  **System.err.print(" 2");**  **} catch (RuntimeException e) {**  **System.err.print(" 3"); // НЕ заходим - есть исключение, но НЕПОДХОДЯЩЕГО ТИПА**  **} finally {**  **System.err.print(" 4"); // заходим всегда**  **}**  **System.err.print(" 5");     // НЕ заходим - выполнение НЕ в норме**  **} catch (Exception e) {**  **System.err.print(" 6");     // не заходим - есть исключение, но НЕПОДХОДЯЩЕГО ТИПА**  **} finally {**  **System.err.print(" 7");     // заходим всегда**  **}**  **System.err.print(" 8");         // не заходим - выполнение НЕ в норме**  **}**  **}**    >> 0 1 4 7  >> RUNTIME EXCEPTION: Exception in thread "main" java.lang.Error |

Мы НЕ заходим в ОБЕ catch-секции (не печатаем "3" и "6"), заходим в обе finally-секции (печатаем "4" и "7") и в обоих случаях НЕ выполняем код ПОСЛЕ finally (не печатаем "5" и "8", так как исключение НЕ остановлено), выполнение метода прерывается по исключению.

**Тесты по всей лекции**

Тест, состоящий из случайных вопросов тестов этой лекции

Some issues occured with quiz randomizer :(

**Лабораторные**

**Лабораторная: ex.try-catch-finally.atomic\_tx**

Это исключение характеризует собой временную неудачу. Значит  
1. ничего не было сделано  
2. попробуйте вызвать метод еще раз (и еще раз, и еще раз, ...)

[?](http://www.golovachcourses.com/exceptions-try-catch-finally/)

|  |  |
| --- | --- |
| 1 | public class TryAgainException extends Exception {} |

Это исключение характеризует полное блокирование счета, значит  
1. ничего не было сделано  
2. нет смысла делать повторные вызовы (получишь то же исключение)

[?](http://www.golovachcourses.com/exceptions-try-catch-finally/)

|  |  |
| --- | --- |
| 1 | public class BlockAccountException extends Exception {} |

Account - это счет отдельного пользователя.  
change(int delta) - метод, меняющий состояние счета на delta (может быть отрицательным)

[?](http://www.golovachcourses.com/exceptions-try-catch-finally/)

|  |  |
| --- | --- |
| 1  2  3 | public interface Account {      public void change(int delta) throws TryAgainException, BlockAccountException;  } |

AccountManager - это та часть системы, которую необходимо реализовать (менеджер счетов).  
Получив массив счетов и массив сумм на изменение (такой же длины) он должен вызвать для всех счетов accounts[k].change(deltas[k]). Если на каком-то счете получили TryAgainException - надо просто повторять операцию, если получили BlockAccountException, то надо "откатить ситуацию назад" (вернуть деньги на счета, с которых уже сняли/начислили). Возвращает true - если получилось перевести деньги на все счета, false - в обратном случае.

[?](http://www.golovachcourses.com/exceptions-try-catch-finally/)

|  |  |
| --- | --- |
| 1  2  3 | public interface AccountManager {      public boolean transfer(Account[] accounts, int[] deltas);  } |

Плохая реализация. "Трусливый" менеджер:  
1) плохо: столкнувшись с исключением сразу сдается  
2) плохо: не пытается вернуть состояние счетов

[?](http://www.golovachcourses.com/exceptions-try-catch-finally/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class AccountManagerImpl implements AccountManager {      public boolean transfer(Account[] accounts, int[] deltas) {          try {              for (int k = 0; k < accounts.length; k++) {                  accounts[k].change(deltas[k]);              }          } catch (Exception e) {              return false;          }          return true;      }  } |

Плохая реализация. "безоткатный" менеджер:  
1) молодец: столкнувшись с TryAgainException не сдается  
2) плохо: столкнувшись с BlockAccountException не пытается вернуть состояние счетов

|  |  |
| --- | --- |
|  | public class AccountManagerImpl implements AccountManager {      public boolean transfer(Account[] accounts, int[] deltas) {          try {              for (int k = 0; k < accounts.length; k++) {                  while (true) {                      try {                          accounts[k].change(deltas[k]);                          break;                      } catch (TryAgainException e) {                          // NOP                      }                  }              }          } catch (BlockAccountException e) {              return false;          }          return true;      }  } |

**Литература**

РАЗДЕЛ НАХОДИТСЯ В РАЗРАБОТКЕ

Шилдт. "Java. Полное руководство". Издание 8  
....Глава 10. Обработка исключений, стр 239-256

Хорстманн, Корнелл. "Java 2. Том I. Основы". Издание 7  
....Глава 11. Исключения и отладка, стр 657-682