**Выдержки из книги “Java. Notes for Professionals”**

**(она есть в moodle – см. JavaForProfessionals.pdf)**

**Exceptions**

Объекты типа java.lang.**Throwable** и его подтипов могут посылаться вверх по стеку вызовов с помошью ключевого слова **throw** и отлавливаться с помощью конструкций **try…catch**, **try…catch…finally** и **try-with-resources**.

**Try-catch with one catch block**

**try** {

doSomething();

} **catch** (SomeException e) {

handle(e);

}

// next statement

**Поведение:**

* Выполняются операторы внутри блока **try;**
* Если при этом не происходит исключений, то управление передается оератору, следующему за **try…catch**;
* Если внутри блока try кидается исключение, то
* Проверяется, приводится ли объект-исключение к типу SomeException;
* Если да, то блок catch «ловит» исключение, то есть:
* Объект-исключение связывается с переменной e;
* Исполняется код блока catch;
* Если этот код кидает исключение, то это новое исключение распространяется вместо исходного; иначе – управление передается следующему за блоком **catch** оператору.
* Если нет, исходное исключение распространяется дальше по стеку (вверх).

**Try-catch with multiple catches**

**try** {

doSomething();

} **catch** (SomeException e) {

handleOneWay(e);

} **catch** (SomeOtherException e) {

handleAnotherWay(e);

}

// next statement

**Поведение:**

Если имеются несколько блоков **catch**, они проверяются по очереди сверху вних до тех пор, пока не найдется подходящий. При этом будет выполнен соответствующий обработчик, как указано выше, и управление передастся оператору, следующему за последним блоком **catch**. Все блоки **catch** после сработавшего будут пропущены – *даже если код обработчика кидает исключение*.

Такая стратегия имеет последствия для случаев, когда исключения не являются независимыми. **Например:**

**try** {

**throw new** RuntimeException("test");

} **catch** (Exception e) {

System.out.println("Exception");

} **catch** (RuntimeException e) {

System.out.println("RuntimeException");

}

Этот код напечатает **Exception**, а не **RuntimeException**. Поэтому более специфические исключения надо отлавливать перед более общими (иначе их специфическая обработка будет пропущена). *Некоторые компиляторы на это дают warning, но это – не ошибка компиляции.*

***ЭТО – неверно: JDK 1.8 дает ошибку компиляции, и код не будет выполнен вообще. Это – к вопросу о важности рельной работы с примерами из учебников...***

**Multi-exception catch blocks (Version ≥ Java SE 7)**

Начиная с JDK7, можно в одном блоке catch обрабатывать список не относящихся друг к другу исключений.

**try** {

doSomething();

} **catch** (SomeException | SomeOtherException e) {

handleSomeException(e);

}

Поведение аналогично случаю с одним exception, где **catch** срабатывает, если exception соответствует хотя бы одному из перечисленных типов. Тут имеется одна тонкость: типом e является «синтетический тип», объединяющий все перечисленные (их наименьший общий супертип). Однако, если exception перевыкидывается снова внутри блока **catch**, то типы exceptions, которые могут быть произведены, - это все те типы, которые были в «объединении». (См. пример MultiCatchSample)

**См. также главу 131: Java Pitfalls – Exception usage**

Засады при использовании exceptions

**Засада №1. Ловля Throwable, Exception, Error или RuntimeException**

Типичный образ мышления неопытного программиста: exceptions – это проблема, которую гнадо поймать как можно раньше. Это приводит к такому коду:

....

**try** {

InputStream is = **new** FileInputStream (fileName);

*// process the input*

} **catch** (Exception ex) {

System.out.println ("Could not open file” + fileName);

}

Проблема в том, что ловиттся больше exceptions, чем кажется (например, - null pointer exception при filename null, что приведет к бессмысленной печати: *Could not open file null*).

**Проблема – в ловле слишком общих исключений (что почти всегда ошибка!):**

* Ловля **Exception** будет ловить все контролируемыеисключения и большинство неконтролируемых.
* Ловля **RuntimeException** будет ловить большинство неконтролитуемых исключений.
* Ловля **Error** словит все неконтролируемые ошибки, которые не надо ловить, так как их нельзя исправить.
* Ловля **Throwable** отловит вооще все.

**Правильно ловить не все, что что можно поймать, а то, что действительно происходит:**

**try** {

InputStream is = **new** FileInputStream (fileName);

*// process the input*

} **catch** (FileNotFoundException ex) {

System.out.println ("Could not open file” + fileName);

}

**Засада №2. Игнорирование или подавление Exception**

Как можно ловить exceptions, не обращая на них внимание при этом? Как правило, это – плохой способ обработки исключений.

Если exception подавляется, ошибка может проявляться в причудливых формах, которые трудно опознать. Ошибку становится труднее найти, так как информация о ней утрачивается.

Иногда – когда это оравдано – можно игнорировать (хорошие IDE это понимают и не дают warning, который дают при полном подавлении):

**try** {

selfie.show();

} **catch** (InterruptedException ignored) { }

**Засада №3. Кидание Throwable, Exception, Error или RuntimeException**

Ловля Throwable, Exception, Error или RuntimeException – это плохо. Но кмдать их еще хуже.

Наличие высокоуровневых (по иерархии) исключений усложняет анализ условий возникновения и обработки ошибок.

**try** {

InputStream is = **new** FileInputStream (someFile);

*// could throw IOException*

...

**if** (somethingBad) {

**throw new** Exception (); ***// WRONG***

}

} **catch** (IOException ex) {

System.err.println ("cannot open ...");

} **catch** (Exception ex) {

System.err.println ("something bad happened"); ***// WRONG***

}

Если мы кидаем Exception, мы должны его ловить, а это – засада № 1(см. выше).

**См. все засады в chapter 131…**