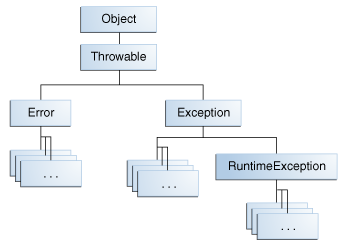
Исключения в Java

Исключениями или исключительными ситуациями (состояниями) называются ошибки, возникшие в программе во время её работы, [1].

Все исключения в Java являются объектами. Поэтому они могут порождаться не только автоматически при возникновении исключительной ситуации, но и создаваться самим разработчиком.

Иерархия классов исключений:



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

* **Exceptions**  являются результатом проблем в программе, которые в принципе решаемы и предсказуемы. Например, произошло деление на ноль в целых числах;
* **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.

В Java все исключения делятся на три типа, [4]:

* **Контролируемые исключения (checked exceptions)**. Представляют собой ошибки, которые можно и нужно обрабатывать в программе, к этому типу относятся исключения типа Throwable, все потомки класса Exception (кроме RuntimeException).

Если в коде есть участок который может бросить checked исключение его обязательно нужно заключить в конструкцию try/catch либо объявить в сигнатуре метода throws чтобы делегировать обработку исключения на уровень выше. Наличие обработки проверяется на этапе компиляции

В любом случае исключение нужно перехватить. В противном случае программа просто не скомпилируется. После обработки контролируемого исключения программа может продолжить работу;

* **Неконтролируемые исключения (unchecked)**, к которым относятся ошибки (Errors). Это условия внешние по отношению к программе, после которых она обычно не может продолжить работу. Например, программа открыла файл, но не может прочитать его из-за неисправности оборудования. В этом случае будет сгенерировано исключение java.io.IOError.

Приложение может перехватить unchecked исключение, но лучше будет вывести stack trace и выйти из программы. Наличие обработки проверяется во время выполнения

* **Исключения времени выполнения (RuntimeExceptions)**. Это проблемы внутренние по отношению к программе, после которых программа не может продолжать свое выполнение. Это исключение относится к **неконтролируемым**. Чаше всего это логические ошибки, неправильное использование API и т. п.

Приложение может перехватывать RuntimeException исключение, но лучше переписать код, чтобы исключить возникновение ошибки.

Обработка исключений

[Обработка исключительных ситуаций](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9) (exception handling) - механизм языков программирования, предназначенный для описания реакции программы на ошибки времени выполнения и другие возможные проблемы (исключения), которые могут возникнуть при выполнении программы и приводят к невозможности (бессмысленности) дальнейшей отработки программой её базового алгоритма, [2], [3].

В Java есть пять ключевых слов для работы с исключениями:

1. **try** - данное ключевое слово используется для отметки начала блока кода, который потенциально может привести к ошибке;
2. **catch** - ключевое слово для отметки начала блока кода, предназначенного для перехвата и обработки исключений;
3. **finally** - ключевое слово для отметки начала блока кода, которое является дополнительным. Этот блок помещается после последнего блока 'catch'. Управление обычно передаётся в блок 'finally' в любом случае. Блок finally очень удобен для закрытия файлов и освобождения любых других ресурсов, захваченных для временного использования в начале выполнения метода.

**finally**-секция не вызывается только если мы прекратили выполнение 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

1. **throw** - служит для генерации исключений;
2. **throws** - ключевое слово, которое прописывается в сигнатуре метода, и обозначающее что метод потенциально может выбросить исключение с указанным типом;

Хотя встроенные исключения Java обрабатывают большинство частых ошибок, можно создать собственные типы исключений для обработки ситуаций, специфичных для пользовательского приложения. Для этого нужно определить подкласс Exception.

Одному блоку **try** может соответствовать сразу **несколько блоков catch** с разными классами исключений.

Последовательность блоков catch должна идти от частного к более общему. В противном случае будет ошибка компиляции. По типу потомку нельзя перехватить тип предок

Что такое stack trace

При работе JVM использует стек вызовов, также, как и в случае с компилируемыми языками типа С++. При возникновении исключения в приложении JVM «раскручивает» стек вызовов от вершины вглубь в поисках обработчика исключения. Если обработчик исключения не будет найден стек будет просмотрен до точки входа, [5].

Для отладки весьма полезно иметь средства просмотра стека вызовов и такие средства предусмотрены в Java на уровне языка.

При «раскрутке» стека JVM заполняет массив объектов StackTraceElement. Доступ к информации можно получить через объект исключения:

* Вывод в консоль e.printStackTrace(System.out);

catch(Exception e) {

e.printStackTrace(System.out);

}

* Доступ к массиву StackTraceElement

catch (Exception e) {

for(StackTraceElement ste : e.getStackTrace())

System.out.println(ste.getMethodName());

}

Можно также получить доступ к массиву StackTraceElement через объект текущего потока:

Thread.currentThread().getStackTrace();

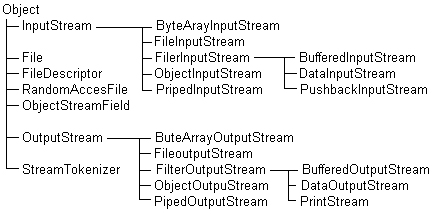
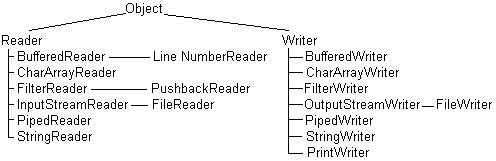
Далее нужно воспользоваться методами StackTraceElement чтобы извлечь нужную информацию.

Потоки ввода / вывода в Java

Обобщенное понятие источника ввода относится к различным способам получения информации: к чтению дискового файла, символов с клавиатуры, либо получению данных из сети. Аналогично, под обобщенным понятием вывода также могут пониматься дисковые файлы, сетевое соединение и т.п.

Эти абстракции дают удобную возможность для работы с вводом-выводом (I/O), не требуя при этом, чтобы каждая часть кода понимала разницу между, скажем, клавиатурой и сетью. В Java эта абстракция называется потоком (stream) и реализована в нескольких классах пакета java.io.

Потоки в Java делятся на:

* Байтовые. Эти потоки предоставляют средства для ввода/вывода байтов. Используются при работе с бинарными данными. На рисунке ниже показана иерархия классов байтовых потоков  
  
* Символьные. Эти потоки предоставляют средства для ввода/вывода символов. Символьные потоки используют кодировку Unicode. На рисунке ниже показана иерархия классов символьных потоков.  
  
* **FileInputStream**. Получает байты из файла в файловой системе. FileInputStream предназначен для чтения потока байтов, не учитывая какую информацию эти байты представляют.
* **FileOutputStream**. Осуществляет запись потока байтов в файл или (FileDescriptor). Доступность файла и возможность его создания зависят от платформы.

1. Исключения и их обработка, <http://kostin.ws/java/java-exceptions.html>
2. Исключения в Java, Часть I (try-catch-finally), <http://habrahabr.ru/company/golovachcourses/blog/223821/>
3. Исключения в Java, <http://www.quizful.net/post/java-exceptions>
4. The Three Kinds of Exceptions, <http://docs.oracle.com/javase/tutorial/essential/exceptions/catchOrDeclare.html>
5. Thinking in Java Fourth Edition, Bruce Eckel, с. 324
6. Java 7- The Complete Reference - 8th Edition, Herbert Schildt, с. 318-320.