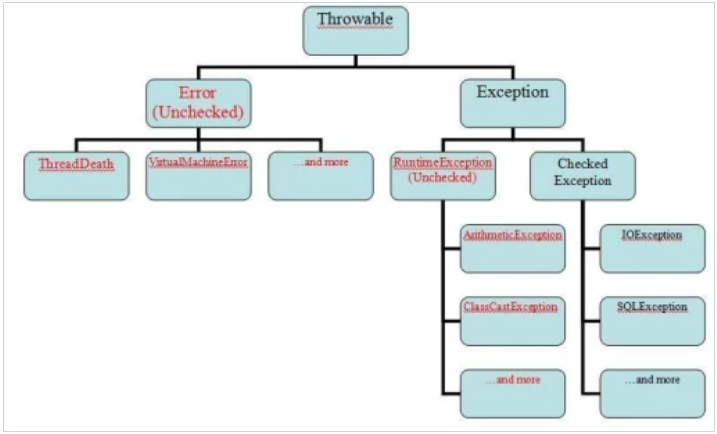
**Исключения Основы**

**Предупреждение о «неприятностях»**

Когда вы **не** планируете обрабатывать исключение в своем методе, но хотите предупредить пользователей метода о возможных исключительных ситуациях — используйте ключевое слово **throws**. Это ключевое слово в сигнатуре метода означает, что при определенных условиях метод, может выбросить исключение. Такое предупреждение является частью интерфейса метода и предоставляет право пользователю на собственный вариант реализации обработчика исключения. После throws мы указываем тип выбрасываемого исключения. Обычно это наследники класса Exception Java. Поскольку Java является объектно-ориентированным языком, все исключения в Java представляют собой объекты.



**Иерархия исключений Java**

При возникновении ошибки в процессе выполнения программы исполняющая среда JVM создает объект нужного типа из иерархии исключений Java – множества возможных исключительных ситуаций, унаследованных от общего «предка» – класса **Throwable**.

**Исключительные ситуации, возникающие в программе, можно разделить на две группы:**

1. Ситуации, при которых восстановление дальнейшей нормальной работы программы невозможно.
2. Восстановление возможно.

**К первой группе относят** ситуации, когда возникают исключения, унаследованные из класса **Error**. Это ошибки, возникающие при выполнении программы в результате сбоя работы JVM, переполнения памяти или сбоя системы. Обычно они свидетельствуют о серьезных проблемах, устранить которые программными средствами невозможно. Такой вид исключений в Java относится к неконтролируемым (unchecked) на стадии компиляции.

К этой группе также относят **RuntimeException** – исключения, наследники класса Exception, генерируемые JVM во время выполнения программы. Часто причиной возникновения их являются ошибки программирования.

**Error** ошибки не обрабатываются. **RuntimeException** исключения также являются неконтролируемыми (unchecked) на стадии компиляции, поэтому написание кода по их обработке не является обязательным.

**Ко второй группе относят** исключительные ситуации **CheckedExeption**, предвидимые еще на стадии написания программы, и для которых должен быть написан код обработки. Такие исключения являются контролируемыми (checked). Основная часть работы разработчика на Java при работе с исключениями – обработка таких ситуаций.

**IOExceptions**

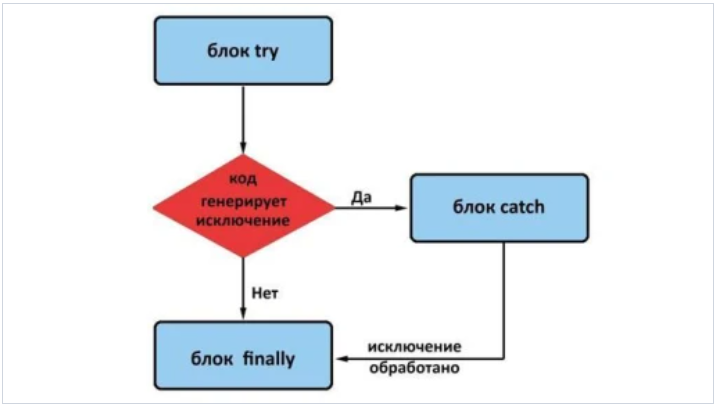
IOExceptions - это исключения ввода-вывода (I / O), которые возникают всякий раз, когда операция ввода-вывода завершается неудачно или интерпретируется. Например, если вы пытаетесь прочитать файл, который не существует, Java выдаст исключение ввода-вывода.

**Создание исключения**

При исполнении программы исключение генерируется JVM или вручную, с помощью оператора **throw**. При этом в памяти создается объект исключения и выполнение основного кода программы прерывается, а обработчик исключений **JVM** пытается найти способ обработать исключение.

**Обработка исключения**

Создание блоков кода, для которых мы предусматриваем обработку исключений в Java, производится в программе с помощью конструкций try{}catch, try{}catch{}finally, try{}finally{}.



При возбуждении исключения в блоке try обработчик исключения ищется в следующем за ним блоке catch. Если в catch есть обработчик данного типа исключения – управление переходит к нему. Если нет, то JVM ищет обработчик этого типа исключения в цепочке вызовов методов до тех пор, пока не будет найден подходящий catch. После выполнения блока catch управление передается в необязательный блок finally. В случае, если подходящий блок catch не найден, JVM останавливает выполнение программы, и выводит стек вызовов методов – stack trace, выполнив перед этим код блока finally при его наличии. Пример обработки исключений:

**public** **class** Print {

**void** print(String s) {

**if** (s == **null**) {

**throw** **new** NullPointerException("Exception: s is null!");

}

System.out.println("Inside method print: " + s);

}

**public** **static** **void** main(String[] args) {

Print print = **new** Print();

List list= Arrays.asList("first step", **null**, "second step");

**for** (String s:list) {

**try** {

print.print(s);

}

**catch** (NullPointerException e) {

System.out.println(e.getMessage());

System.out.println("Exception was processed. Program continues");

}

**finally** {

System.out.println("Inside bloсk finally");

}

System.out.println("Go program....");

System.out.println("-----------------");

} } }

Результаты работы метода main:

Inside method print: first step

Inside bloсk **finally**

Go program....

-----------------

Exception: s is **null**!

Exception was processed. Program continues

Inside bloсk **finally**

Go program....

-----------------

Inside method print: second step

Inside bloсk **finally**

Go program....

-----------------

Блок finally обычно используется для того, чтобы закрыть открытые в блоке try потоки или освободить ресурсы. Однако при написании программы не всегда возможно уследить за закрытием всех ресурсов. Для облегчения нашей жизни разработчики Java предложили нам конструкцию try-with-resources, которая автоматически закрывает ресурсы, открытые в блоке try. Наш первый пример можно переписать так с помощью try-with-resources:

**public** String input() **throws** MyException {

String s = **null**;

**try**(BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(System.in))){

s = reader.readLine();

} **catch** (IOException e) {

System.out.println(e.getMessage());

}

**if** (s.equals("")){

**throw** **new** MyException ("String can not be empty!");

}

**return** s;

}

Благодаря возможностям Java, начиная с версии 7, мы также можем объединять перехват разнотипных исключений в одном блоке, делая код более компактным и читабельным. Например:

**public** String input() {

String s = **null**;

**try** (BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(System.in))){

s = reader.readLine();

**if** (s.equals("")) {

**throw** **new** MyException("String can not be empty!");

}

} **catch** (IOException | MyException e) {

System.out.println(e.getMessage());

}

**return** s;

}