Обработка исключений

<https://docs.oracle.com/javase/tutorial/essential/exceptions/>

Исключением является событие, происходящее во время выполнения программы, которое нарушает нормальный поток инструкций.

Когда в методе возникает ошибка, метод создает объект и передает его системе времени выполнения. Объект, называемый объектом исключения, содержит информацию об ошибке, в том числе ее тип и состояние программы в момент ее возникновения. Создание объекта исключения и передача его системе выполнения называется созданием исключения.

После того, как метод выдает исключение, система выполнения пытается найти что-то для его обработки. Набор возможных «что-то» для обработки исключения — это упорядоченный список методов, которые были вызваны, чтобы добраться до метода, в котором произошла ошибка. Список методов известен как стек вызовов.

Система выполнения ищет в стеке вызовов метод, содержащий блок кода, способный обработать исключение. Этот блок кода называется обработчиком исключений. Поиск начинается с метода, в котором возникла ошибка, и продолжается по стеку вызовов в порядке, обратном тому, в котором были вызваны методы. Когда соответствующий обработчик найден, система выполнения передает исключение обработчику. Обработчик исключения считается подходящим, если тип создаваемого объекта исключения соответствует типу, который может быть обработан обработчиком.

Говорят, что выбранный обработчик исключений перехватывает исключение.

Если исполняющая система исчерпывающе просматривает все методы в стеке вызовов, и не находит подходящего обработчика исключений, исполняющая система (и, следовательно, программа) завершает работу.

Класс Exception.

<http://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html>

В языке Java для обработки исключительных ситуаций предусмотрен ряд классов, а также подклассов. На верхнем уровне находится Trowable , подклассы его делятся на два подкласса: Exception и Error.

Ошибки типа Error возникают в самой исполняющей системе Java связаны как правило с аварийными ситуациями в самой системе, на которые пользователь влияния оказать не может, перехватить такое исключение, обработать его и направить исполнение программы по нормальному потоку инструкций невозможно, т.к. ошибки в действительности оказываються критичными и не совместимыми с дальнейшим нормальных ходом исполнения программы.

Ошибки типа Exception служат для случаев , которые должен перехватывать код прикладной программы. При создании собственных случаев исключений используют подклассы этого класса Exception.

Суперклас Exception , расширяют еще два подкласса: RuntimeException и IOException.

Это так называемые проверяемые и непроверяемые компилятором исключения (checked and unchecked).

Т.е. RuntimeException это исключения которые могут возникнуть в ходе выполнения программы и они относятся к unchecked , т.е. мы их не обязаны отработать.

А IOException это исключения checked т.е. те которые видит компилятор на уровне компиляции и говорит нам что тут может быть такая то ошибка возникнуть – посмотри на неё и отработай её. Пропустить такое исключение не отработав его или не вытолкнув выше в код - нам не получится. Компилятор не даст проигнорировать.

В свою очередь классы RuntimeException и IOException делятся и имеют свои множественные специализированные подклассы которые в конце иерархии указывают на конкретное исключение.

try - catch

<https://docs.oracle.com/javase/tutorial/essential/exceptions/handling.html>

Существуют три блока: try, catch и finally — которые служат для написания обработчика исключений.

Блок: try — в него заключается потенциально опасный код в котором могут возникнуть исключения, в этом блоке будут следить за появление исключений, и если оно появится, блок try схватит его и передаст дальше.

Блок: catch — перехватывает схваченное исключение и обрабатывает его, в этом блоке нужно поместить то исключение обработать которое мы хотим. Если его не будет в этом блоке, то программа передаст исключение выше, и если не найдет и выше обработчика нужного исключения - то сама его обработает с аварийным завершением работы программы.

Блок: finally — после блока: try, catch, можно указать блок : finally , он не является обязательным, но в некоторых случаях может быть полезен. В нем размещаются операторы которые должны выполняться обязательно после завершения блока: try. Блок: finally выполняется независимо от того , было ли поймано исключение или нет, и даже если ни один оператор catch не содержит нужного исключения все равно блок : finally – должен отработать.

С появлением Java SE 7 появился оператор try-with-resources. Этот оператор try-with-resources особенно подходит для ситуаций, в которых используются ресурсы Closeable, такие как потоки.

Допустим был объявлен сканер – его объявление можно записать в блоке оператора try-with-resources после выполнения блоков: try, catch и finally – поток открытый сканером будет закрыт.

Оператор try-with-resources — это оператор try, который объявляет один или несколько ресурсов. Ресурс — это объект, который должен быть закрыт после того, как программа закончит работу с ним. Оператор try-with-resources гарантирует, что каждый ресурс будет закрыт в конце оператора. В качестве ресурса может использоваться любой объект, реализующий java.lang.AutoCloseable, включая все объекты, реализующие java.io.Closeable.