Exception

Исключительная ситуация - неожиданная ситуация, проблема возникшая во время выполнения программы. Например:

1. Файл, к которому обращается программа, не найден.
2. Сервер прислал неожиданный ответ.
3. Пользователь ввел некорректные данные.
4. Сетевое соединение с сервером было прервано.
   И т.д. и т.п.

Exception (англ. исключение) — это механизм язык, позволяющий обработать *исключительную ситуацию* и восстановить *нормальный поток выполнения программы.*

С точки зрения реализации в Java, Exception - это объект класса Throwable или одного из потомков Throwable, созданный программой при возникновение *исключительной ситуацыи*.

Объект exception может быть создан как методом написанным вами, так и любым сторонним методом. Обычно, в документации к методу перечислены случаи, когда и при каких обстоятельствах метод вызывает exception.

Exception - это возможность определить поведение программы при возникновение *исключительной ситуации,* например: сообщить сведения о причинах или использовать другие данные, или попытаться повторить операцию и т.д. и т.п.

Обработка исключений (Exceptions Handling)

В Java есть пять ключевых слов для работы с исключениями:

1. **try** — начало блока кода, который потенциально может сформировать исключение, привести к exception.
2. **catch** — ключевое слово, для отметки блока кода, предназначенного для обработки исключений (еще говорят для *перехвата* исключения)
3. **finally** — ключевое слово для отметки начала блока кода, который должен быть выполнен в любом случае, вне зависимости произошла *исключительная ситуация* или нет.
4. **throw** — позволяет сгенерировать новый exception (*выкинуть исключение)*.
5. **throws** — ключевое слово, которое прописывается в сигнатуре метода, и обозначающее что метод который потенциально может выбросить исключение с указанным типом.

Конструкция try-catch-finally

Конструкция try-catch-finally предназначена для перехвата и обработки исключений.

```
try {
      // блок try (1): код, который может выкинуть Exception
       riskyMethod();
} catch (Exception e) {
     // блок catch (2):код обработки Exception
     System.err.print("Что то пошло не так!");
      e.printStackTrace(); // выводит полную информацию об исключении
} finally {
      // блок finally (3): код, который выполняем в любом случае
      System.out.print("сохраним критические данные");
}
```

В блок кода (1) `try` мы помещаем потенциально опасный код, код который может сформировать Exception или, как еще говорят “*выкинуть исключение*”. Если при выполнение кода блока try возникает exception, программа прерывает *нормальный поток выполнения* и управление моментально передается в соответствующий блок (2) `catch`.

Блок (2) `catch` выполняется только в случае возникновения *исключительной ситуации* при выполнении блока try. Иными словами, этот блок должен содержать код, который будет выполнен, если в блоке try возник Exception, соответствующего типа. При этом, сформированный системой объект exception будет передан в переменную, указанную в параметрах блока catch (в примере выше это переменная e). Таким образом, через методы этой переменной мы можем получить всю необходимую информацию о произошедшей *исключительной ситуации*.

Блок (3) `finally` будет выполнен в любом случае. Блок finally не является обязательным. Обычно в finally помещают критично-важный код, например освобождение занятых ресурсов, сохранение состояния и т.д. **Внимание:** секция finally выполняется до выхода из метода, т.е. до выполнения return в секции try или, например, в секции catch

После обработки exception т.е. выполнения блоков catch и finally программа продолжит нормальный поток выполнения и выполнит все, что находится следом за try-catch-finally.

Каждому оператору try требуется хотя бы один из операторов catch или finally. Одному оператору try, может соответствовать несколько блоков catch для обработки разных типов Exception:

```
try {
} catch (FileNotFoundException|SQLException e) {
     // блок catch (1): код обработки FileNotFoundException или SQLException
} catch (RuntimeException e) {
     // блок catch (2): код обработки RuntimeException
} catch (Exception e) {
     // блок catch (3): код обработки FileNotFoundException
} finally {
      // блок finally выполняем в любом случае
}
```

В данном примере:

- в случае возникновение FileNotFoundException или SQLException будет выполнен блок catch (1)
- в случае возникновение RuntimeException будет выполнен блок catch (2)
- все остальные Exception вызовут выполнение блока catch (3)

Важно: секции catch обрабатываются последовательно, сверху вниз. Выполняется первая подходящая по типу секция. Таким образом секции должны идти от частных (дочерних) типов exception к более общим (родительским) типам (см. раздел “Иерархия исключений”). Так, блок обрабатывающий самый родительский класс `catch (Exception e)` должен быть всегда последний.

Оператор throw

Оператор throw позволяет создать exception или, как еще говорят выкинуть исключение:

```
if (list == null) {
   throw new IllegalArgumentException("Отсутствуют данные");
}
```

В качестве параметра для оператора throw может быть указан любой, наш или стандартный, Throwable объект, т.е. объект класса Throwable или одного из его потомков (см.раздел “Иерархия исключений”). На практике собственные классы Exception создают наследуя от класса Exception или от класса RuntimeException (см.раздел “Проверяемые и непроверяемые исключения”)

Любой не обработанный Exception немедленно прерывает *нормальный поток выполнения программы*. Exception может быть обработан (try-catch) или передан для обработки вызывающему методу. После обработки будет продолжен *нормальный поток выполнения программы.*

Если метод не обрабатывают возникший exception самостоятельно, выполнение метода моментально прерывается, Exception и управление передается точку вызова, т.е. в метод, который вызвал “ошибочный” код еще говорят, “метод бросает exception”. Если и там Exception не будет обработан, то будет прервано выполнение и родительского метода, и Exception будет проброшен на уровень выше. И так далее, до первого встреченного блока try-catch или до полного прерывания программы, ну т.е. прерывания самого первого метода main().

Например, допустим в нашей программе:

метод main() вызывает метод a(),
метод a() вызывает метод b(),
метод b() вызывает метод c(),
а в методе c() формируется exception

тогда:

- если метод c() не обработает exception, программа прервет выполнение метод c() и передаст exception и управление методу b(). Говорят: “*метод c() пробрасывает исключение*”.
- если в методе b() exception не будет обработан, программа прервет выполнение метод b() и передаст exception и управление методу a()
- если в методе a() exception не будет обработан, программа прервет выполнение метод a() и передаст exception и управление методу main()
- если и в методе main() exception не будет обработан, программа прервет выполнение и передаст exception и управление системному обработчику JVM, который выведет сообщение об exception на экран.

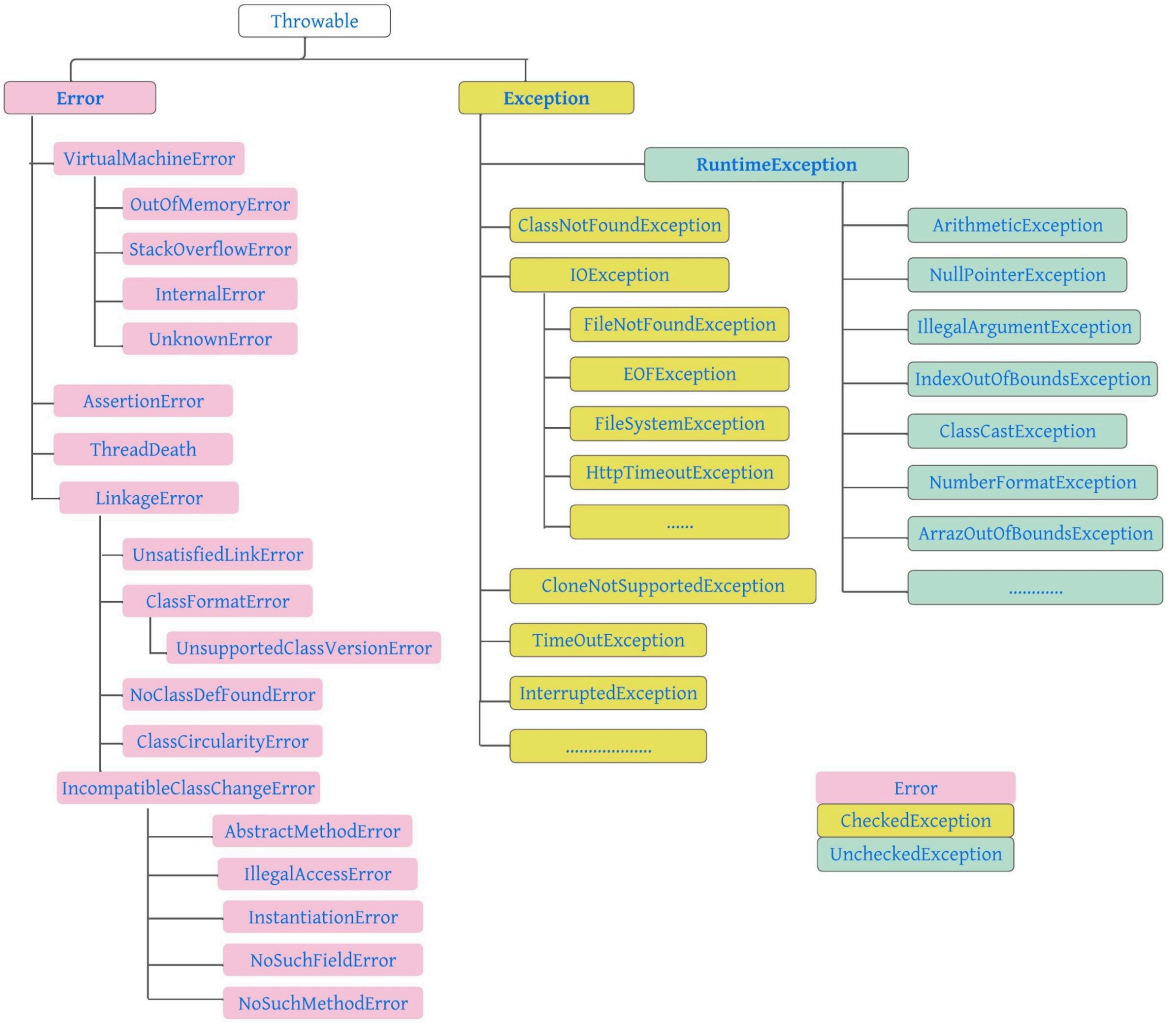
Оператор throws

Если метод *бросает исключение*, т.е. код метода может вызвать exception, но exception не обрабатывается непосредственно в методе , в сигнатуре метода мы должны указать ключевое слово throws и типы бросаемых Exception:

```
public static void riskyMethod() throws IOException,SQLException{
}
```

тем самым как бы предупреждая вызывающий код, что данный метод не безопасен и может сформировать exception-ы. Если класс exception относится к непроверяемым (наследует от RuntimeException или его потомков) секцию throws можно не указывать.

Иерархия исключений



Все исключения в Java имеют общего предка — класс Throwable. Его потомками являются подклассы Exception и Error.

Исключения (Exceptions) являются результатом проблем в программе, которые в принципе решаемы и предсказуемы. Например, файл не доступен для записи.

Ошибки (Errors) представляют собой более серьезные проблемы, которые, согласно спецификации Java, не следует пытаться обрабатывать в собственной программе, поскольку они связаны с проблемами уровня JVM. Например, закончилась память, доступная виртуальной машине. Программа никак не сможет обеспечить для JVM.

Все исключения, наследники класса RuntimeException являются непроверяемыми.

Проверяемые (checked) и непроверяемые (unchecked) исключения

Все исключения в Java можно разделить на два вида: проверяемые (checked) и непроверяемые (unchecked) исключения. К непроверяемым (unchecked) исключениям относятся все классы наследники RuntimeException. Все остальные наследники Throwable - проверяемые (checked).

Все проверяемые (checked) исключения **должны быть** либо обработаны в явном виде с помощью try-catch, либо переданы в вызывающий метод. В последнем случае, метод бросающий исключение **обязан сообщить об этом** в сигнатуре:

```
public static void riskyMethod() throws IOException,SQLException{
}
```

Непроверяемые (Unchecked) исключения также могут быть обработаны с помощью try-catch, либо будут переданы в вызывающий метод. В последнем случае, метод бросающий исключение **не обязан** ничего указывать в сигнатуре (но может):

```
public static void riskyMethod2(){
        throw new RuntimeException()
}
```

В остальном поведение проверяемых (checked) и непроверяемых (unchecked) exception не отличается друг от друга:

- Если исключительная ситуация произошла будет сформирован exception.
- Пока exception не обработан, нормальный поток выполнения программы прерывается.
- Exception может быть пойман и обработан или выкинут в вызывающий метод

Создание собственных исключений

Любой пользовательский класс являющийся наследником Throwable может быть использован как исключение в Java т.е. может быть использован в try…catch, throw или throws.
В примере ниже создается пользовательское исключение `DataMissingException`, которое является подклассом `Exception`.

```
public class DataMissingException extends Exception {
    public DataMissingException(String message) {
        super(message);
    }
}
```

Данное исключение будет проверяемым т.к. наследует от Exception. Для создания непроверяемого исключения достаточно унаследовать от класса `RuntimeException`

В большинстве случаев собственные исключения это очень простые классы, наподобие класса приведенного выше. Само имя класса несет информацию о природе исключения. Но не стоит забывать, что исключение - это класс Java со всеми возможностями классов Мы можем добавить в класс-исключение свои поля, например для передачи дополнительной информации, методы, конструкторы и т.д.