**Exception**

**Exception –** это ошибка возникающая во время выполнения программы.

**Try** - определяет блок кода в котором может произойти исключение.

**Catch** – определяет блок кода в котором будет обработано исключение.

**Finally** – блок не обязательный, но выполняется в любом случае.

Можно создать следующе конструкции:

try {} catch() {}

try {} catch() {}finally{}

try {} finally{}

Блок try требует наличия либо catch либо finally вместе с собой.

Ключевые слова:

**throw** – создаёт исключение (throw new MyException ("наш текст ошибки");).

**throws** – используем в сигнатуре метода, говорим что данный метод может бросить какое то исключение.

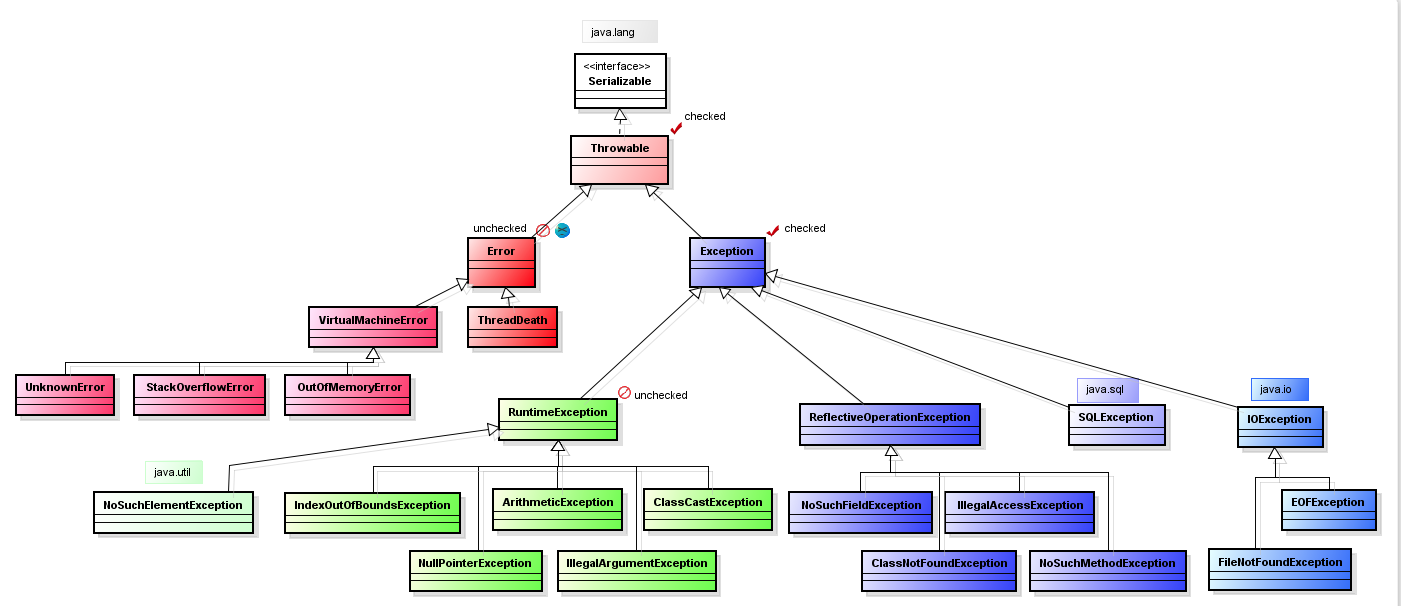
Исключения делятся на два типа:

Checked Exception (Compile time exception) – исключительные случаи в работе программы, обрабатываем и предвидим. Проверка на данные исключения происходит в момент компиляции. Перехват исключений в блоке **catch** происходит в момент выполнения программы.

Unchecked Exception (Runtime exception) – ошибки в программе, их нужно избегать а не обрабатывать!

Если в сигнатуре метода указываем **throws** – значит сообщаем о том что данный метод может выбросить определённое исключение, и способ его обработки предоставляем пользователю метода (как бы “пугаем”). Указывать в сигнатуре метода после **throws**, или блоке **catch** - можем только Throwable и его наследников. Если в блоке **try** было выброшено исключение – то всё что ниже его до конца блока try не выполняется! Переходит если отлавливается исключение в блок **catch** а затем в блок **finally**.

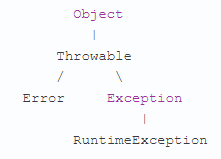
**Все исключения в Java представляют собой объекты.**



Если записываю в System.out то сперва данные записываются в буффер, но если использую System.err то данные выводятся сразу на консоль. Вот почему stacktrace необработанного исключения вылетает в консоль быстрее.

Если метод должен что то вернуть – и в нём сработало исключение, то это освобождает наш метод от возврата чего либо (можем ничего не возвращать). Можем перебить код в **try** – кодом из блока **finally** (смотри заметки).

Общая иерархия исключений:



Как видно Object является предком Throwable – но object не является исключением.

Exception Error – ошибки уровня JVM их не нужно пытаться обрабатывать (закончилась память и тд).

Когда несколько **catch** – нельзя ставить EX **потомка (IOException** к примеру**)** после EX **предка (Exception** к примеру) т.к. до потомка программа не дойдёт. JVM идёт сверху в низ и заходит в тот **catch** в котором указано ваше исключение или его потомок.

Секция **finally** выполняется в любом случае – не зависимо от результата выполнения блока **try**, даже если в нём вызвали return. НЕ выполнится если мы “прибили JVM” – используя в try: System.exit(42) или Runtime.getRuntime().exit(42).

Можно в **throws** передать **unchecked exception** – но это не обязательно и служит для того что бы указать человеку который будет работать с кодом, что данный метод может выбросить данное исключение. Оно не обязывает его как то обрабатывать.

Можно пугать EX родителя (в сигнатуру метода передавать) – а выбрасывать EX потомка. Можно выбрасывать 2 EX потомка и пугать одним общим родителем этих двух EX.

Нельзя выбрасывать EX родителя и пытаться его словить EX потомка – ОШИБКА. Так же нельзя выбрасывать EX брата (Exception) и пытаться его обработать EX (Error) – ОШИБКА и наоборот.

Если обработка EX произошла внутри метода – то можно не пугать этим EX. Если 2 EX внутри метода и один обработан а другой нет – то тот что не обработан тем и пугаем.

Если переопределяем метод и пытаемся у переопределённого метода расширить Exception – указать EX более высоко ранга то происходит ошибка, так как можно только **сужать** тип.

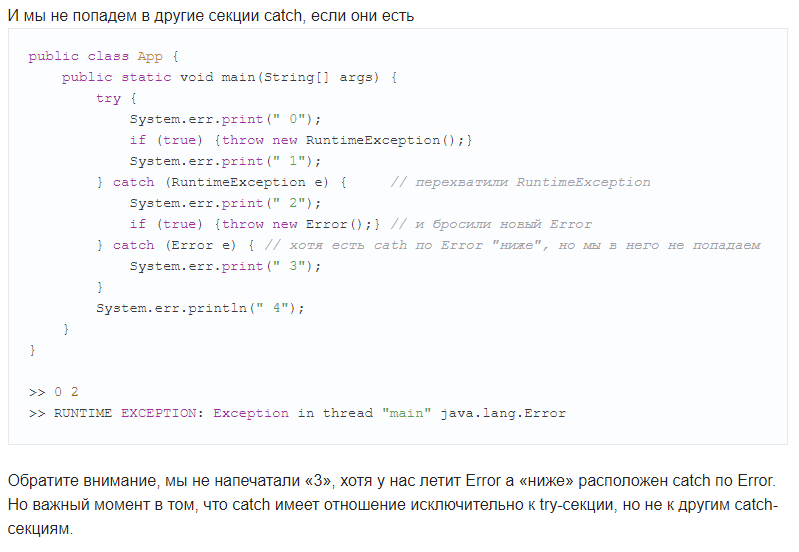
Есть класс А, класс B наследуемый от А, в каждом классе есть метод f(). Если А.f() **throws** Exception а метод класса B.f() **throws** Throwable и при этом мы предаём класс А в класс С и в нём обрабатываем исключекние в блоке catch (Exception) – то всё хорошо. Но если в этот класс передадим B то в классе С произойдёт ошибка из-за попытки дочернего метода расширить EX родительского метода.

**Заметки:**

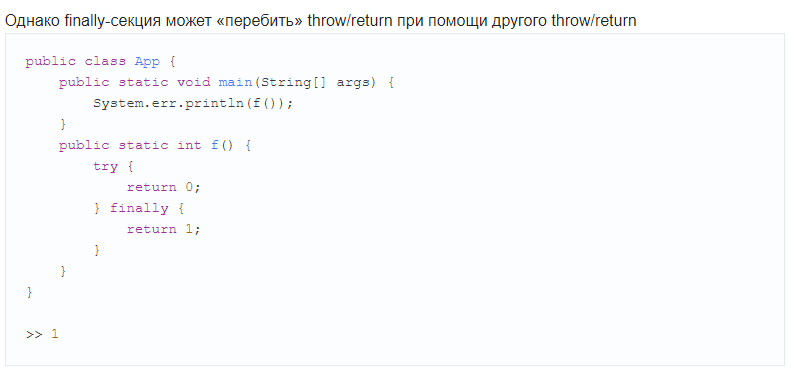
**#1**



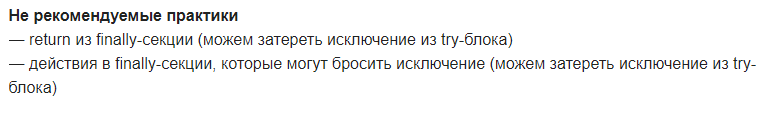
**#2**



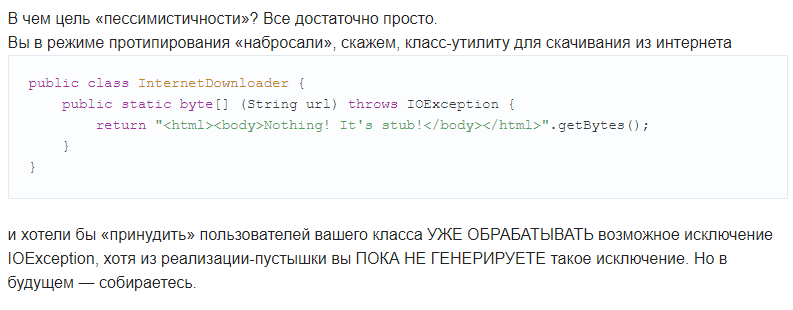
**#3**



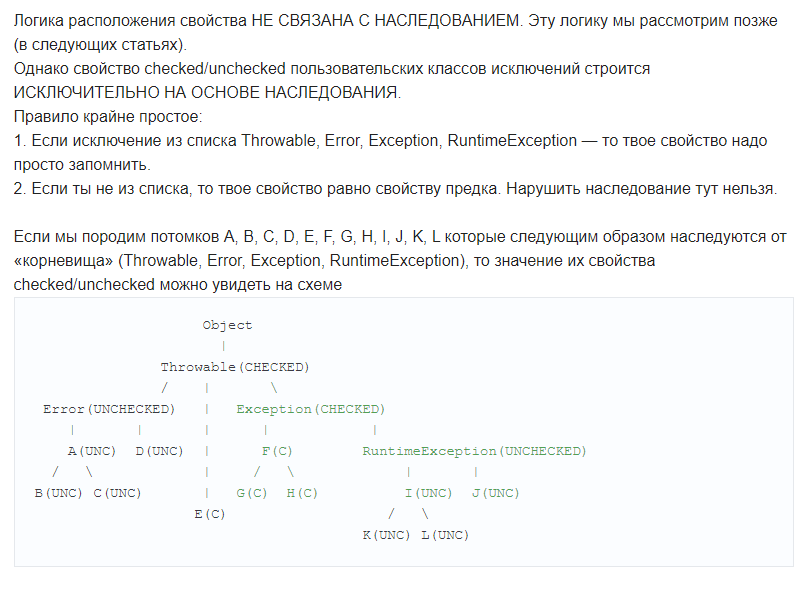
 ****



**#4**



**#5**



**#6**

Запомни – что если класс implements от интерфейса Closeable – следовательно Java сама закроет их (классы,потоки). Можно создавать текущий обьект класса в блоке try-with-resources (Java 7):

public class ReaderObject {  
 public static void main(String[] args) {  
 final File file = new File("C:/Users/anpoliakov/Desktop/testSerial/people.bin");  
  
 try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file))) {  
 Persone [] persones = (Persone[]) ois.readObject();  
 System.*out*.println(Arrays.*toString*(persones));  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
}

Отличная статья для понимания: <https://habr.com/ru/post/178405/>