При обработке исключений в Java применяются следующие **ключевые слова**: — **try** – служит для определения блока кода, в котором может произойти исключение; Могут быть вложенными, как в другой try, так и в catch, finally.

— **catch** – необходим для определения блока кода, где происходит **обработка исключения**;

В одном блоке catch можно обрабатывать несколько исключений



Любо использовать несколько блоков catch, если различные исключения должны обрабатываться по разному.

— **finally** – применяется для определения блока кода, являющегося необязательным, однако при его наличии он выполняется в любом случае вне зависимости от того произошло ли исключение в блоке try.

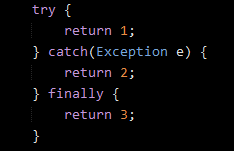
Кроме того:

1. Для возбуждения исключения используем **throw**.

*Throw e* пробросить возникнувшее исключение на уровень выше, throw new Exception(e) – создать новое исключение, где в качестве причины указать e.

При использовании второго варианта мы не даем исключениям низкого уровня пробросится на верхний уровень. Это уменьшает привязанность модулей верхнего уровня от нижних.

1. Для предупреждения в сигнатуре методов о том, что метод может выбросить исключение, применяем **throws**.



Данный блок кода вернет 3. То есть finally выполняется даже после возврата с помощью return.

**Try with resourse**

Данная конструкция, которая появилась в Java 7, позволяет использовать блок *try-catch* не заботясь о закрытии ресурсов, используемых в данном сегменте кода. Ресурсы объявляются в скобках сразу после try, а компилятор уже сам неявно создаёт секцию finally, в которой и происходит освобождение занятых в блоке ресурсов. Под ресурсами подразумеваются сущности, реализующие интерфейс java.lang.Autocloseable. Он имеет метод close, который освободит ресурсы не зависимо от того, были ли исключения.

Общий вид конструкции:

try(/\*объявление ресурсов\*/) {

//...

} catch(Exception ex) {

//...

} finally {

//...

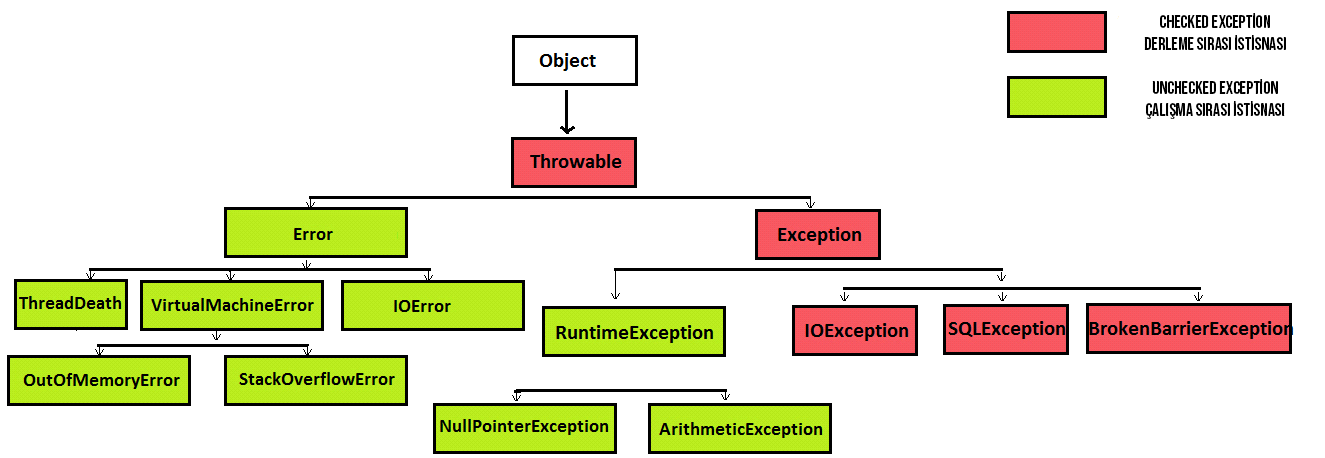
}

Стоит заметить, что блоки catch и явный finally выполняются уже после того, как закрываются ресурсы в неявном finally.

Является заменой закрытию ресурса в блоке finaly. Проблема следующая. Метод close() может сгенерировать исключение. И если при этом основной код работы с ресурсом тоже выбросит исключение, то оно перезатрется исключением из close(). Информация об исходной ошибке пропадёт: мы никогда не узнаем, что было причиной исходного исключения.

При использовании Try with resourse, Если исключение будет выброшено в основном коде и в методе close(), то приоритетнее будет первое исключение, а второе исключение будет подавлено, но информация о нем сохранится (с помощью метода Throwable.addSuppressed(Throwable exception), который вызывается неявно Java компилятором):

**Иерархия исключений**



Исключительные ситуации, которые возникают в программе, делят на 2 группы:

1. Ситуации, при которых восстановление нормальной дальнейшей работы невозможно.

2. Ситуации с возможностью восстановления.

К первой группе можно отнести случаи, при которых возникают исключения, которые унаследованы из класса **Error**. Это ошибки, возникающие во время выполнения программы при сбое работы **Java Virtual Machine**, переполнении памяти либо сбое системы, а также **RuntimeException**. Такие ошибки устранить программным способом практически невозможно.

Это **unchecked** исключения. Остальные классы исключений **checked,** и их нужно обработать в блоке try/catch.

Unchecked исключения, как правило, ошибки программы, которые при правильном кодировании возникать не должны.

Объекты исключений создаются JVM при возникновении исключений.

**Цепочки исключений**

Цепочки исключений – средство, позволяющее связывать одно исключение с другим, чтобы описывать в последнем причину появления первого. Конструкторы



Раньше для обработки ошибок использовался возврат кода в зависимости от успеха либо не удачи. Преимущества исключений в следующем:

* Исключения нельзя игнорировать, разработчик должен их обработать.
* Исключения имеют большую информационную нагрузку, чем код ошибки.