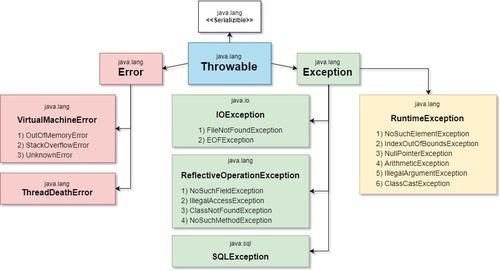
**Исключения –** механизм обработки ошибок и неожиданного поведения программы. Является механизмом управления потоком – при возникновении исключительной ситуации управление передается от обычного потока исполнения в специальные блоки для обработки этой ошибки.

Иерархия:



Throwable – суперкласс для всех исключений – только классы, имеющие этот тип могут быть использованы в конструкциях языка, которые связаны с обработкой исключений (throw, catch. throws).  
Имеет двух наследников Error и Exception:

**Error** – данный класс исключений описывает ошибки, которые указывают на сбои в JVM или на критические ошибки (нехватка ресурсов), которые не нужно обрабатывать по ходу работы приложения. Такие ошибки невозможно устранить программно по ходу работы приложения, что делает непрактичным их обработку с помощью try-catch. Непроверяемые, причина та же – попытки восстановления работы приложения при возникновении такой ошибки неэффективны и небезопасны.  
Примеры: OutOfMemoryError, StackOverflowError, VerifyError и др.

**Exception** – является суперклассом для всех исключений, которые не являются критическими ошибками системы, большинство подклассов являются проверяемыми исключениями, за исключением RuntimeException. Включает исключения, которые возникают в процессе работы программы с «внешним миром», то есть ошибки, которые не зависят от самого приложения.

**RuntimeException** – класс исключений, которые сигнализируют об ошибке в логике работы программы, то есть ошибки которые могут быть обработаны и исправлены программно в ходе работы приложения. Является непроверяемым исключением.

**Checked и Unchecked Исключения**:

Checked – требуют обработки с помощью блока try-catch или явной сигнализации в сигнатуре метода о выбрасываемом исключении. Checked исключениями являются прямые наследники Throwable (если самому наследоваться) все исключения иерархии Exception (кроме Runtime) и соответственно его наследники.

Unchecked – не требуют обработки try-catch или сигнализации в сигнатуре (это можно делать, но это работает, скорее, как аннотация). Unchecked исключениями являются Error и RuntimeException.

**Смысл исключений:**

Checked исключения связаны с ошибками при взаимодействии приложения с «внешним миром» и действуют с точки зрения большей безопасности (если что-то может выбросить исключение, то оно это сделает), при этом накладывается обязательство на обработку или проброс выше (некая форма контракта), что в свою очередь ведет к разрастанию кода.

Unchecked исключения не накладывают таких обязательств, что приводит к меньшей надежности.

**Обработать или прокинуть дальше?**

Если на текущем уровне абстракции невозможно повлиять на это исключение, то его нужно пробрасывать дальше.

**Ключевые слова, конструкции, механика:**

**Ключевые слова:**

*Throw, throws, блок try-catch-finally* – ключевые слова, которые работают только с наследниками Throwable.

*Throws* – располагается в сигнатуре метода, сигнализирует о том, что метод может выкинуть исключение. Если указано Checked исключение, то в вызывающем методе требуется обработка или проброс выше по стеку, если Unchecked, то подобные действия не требуются (маркер).

*Throw* – оператор, осуществляющий проброс исключения, работает по механике return, но для исключений.

***Try-catch-finally блок****:* Механизм управления потоком, направленный на работу с исключениями.

Try – блок кода, который в теории может привести к ошибке. Если в блоке выбрасывается исключение, то исполнение блока завершается и управление передается блоку catch.

Catch – блок кода для перехвата и обработки исключений, исполняемый только в случае возникновения ошибки в блоке try. Специфицируется под перехват определенных типов исключений.  
*Сценарии возникновения ошибки:*

Блок try выбросил исключение – выброшенное исключение соотносится с типом исключений, который обрабатывает блок catch. В случае соответствия выполняется блок catch и исключение считается обработанным, и программа исполняется в штатном режиме (имеется в виду код после блока – код после ошибки в try не исполняется). В случае несоответствия исключение пробрасывается выше по стеку и дальнейший код метода не исполняется.

Блок try не выбросил исключение – блок catch не исполняется.

Особенности блоков: Блоков catch может не быть (тогда должен быть finally) или присутствовать несколько, в таком случае при любой ситуации будет исполнен 1 блок catch. Блоки catch полиморфны – перехватывают исключения объявленного типа и всех его наследников, поэтому при объявлении нескольких блоков базовые исключения должны располагаться ниже «частных», в ином случае частные блоки будут недосягаемы, т.к. базовый тип перехватит все исключения.   
С Java 7 можно добавлять несколько исключений в один catch, используется если необходимый к исполнению код одинаков для нескольких типов исключений. Таким образом нельзя указывать исключения из одной иерархии (та же причина, что и с несколькими блоками). catch(IOException | SQLException ex)

Finally – Блок кода, исполняемый независимо от того, было выброшено исключение или нет. За исключением случаев, если сам блок выбрасывает исключение, тогда код ниже не исполнится, вызов метода System.exit(0) в блоке try. В случае, если блок объявлен в потоке демоне, так как приложение завершает свою работу не учитывая работу потоков демонов. Чаще всего блок используется для освобождения ресурсов памяти.

Каждый блок try-catch-finally имеет свою область видимости– переменные в этих блоках не видны «внешнему коду», переменные в одном из блоков не видны в другом (как и все блоки кода).

*Затирание исключений:*

Finally имеет «последнее слово» в работе блока, он способен изменять результат исполнения.

Примеры:  
Блок finally затирает исключение значением 1.

public static int i() {  
 try {  
 throw new RuntimeException();  
 } finally {  
 return 1;  
}

Затирание значения 0 исключением

public static int f() {  
 try {  
 return 0;  
 } finally {  
 throw new RuntimeException();

Затирание значений:

Блок finally затрет возвращаемое блоком try значение.

public static int foo() {  
 try {  
 return 1;  
 } finally {  
 return 0;  
 }  
}

В данном примере вернется 1, т.к. return вычисляемый оператор и сохраняет свое состояние и изменение переменной x блоком finally не повлияет на возвращаемый результат, т.к. вычисление произошло до присвоения.

public static int foo() {  
 int x = 0;  
 try {  
 return 1;  
 } finally {  
 x = 2;  
 }  
}

**Try-with-resources -** Механизм обработки исключений в Java, в котором неявно вызывается блок finally и происходит высвобождение ресурсов с помощью метода close(), независимо от того, было выброшено исключение или нет. Можно использовать только классы наследники AutoClosable. Появился в Java 7.

До Java 7 использование классов, которые требуют ресурсы выглядело так:

Scanner scanner = null;//Создается вне блока try из-за области видимости   
try {   
 scanner = new Scanner(new File("test.txt"));  
 while (scanner.hasNext()) {  
 System.*out*.println(scanner.nextLine());  
 }  
} catch (FileNotFoundException e) {  
 e.printStackTrace();  
} finally {  
 if (scanner != null) { //Необходимая проверка, если scanner == null,   
 scanner.close(); // то метод выбросит ошибку и ресурсы не освободятся

Код с try-with-resources:

try (Scanner scanner = new Scanner(new File("test.txt"));){   
 while (scanner.hasNext()) {  
 System.*out*.println(scanner.nextLine());  
 }  
} catch (FileNotFoundException e) {  
 e.printStackTrace();

Открывать можно сразу несколько ресурсов, закрыты они будут в обратном порядке во избежание проблем с зависимостями ресурсов (Например, есть два ресурса: поток A и поток B, и поток A зависит от потока B, то при закрытии ресурсов в неправильном порядке (сначала B, затем A) поток A может попытаться обратиться к ресурсу B, который уже закрыт, что приведет к ошибкам.).

Открытые ресурсы в блоке try объявляются как final, их нельзя изменить или присвоить им другую ссылку, с Java 9 возможно использование ранее созданных объектов (должны быть final или Effectively final). Объекты, объявленные в операторе try-with-resources также имеют ограниченную область видимости.  
Если блок finally не объявляется в конструкции try-with-resources, то в случае, если метод close() какого-либо ресурса выбрасывает исключение, то оно добавляется как suppressed к основному исключению.