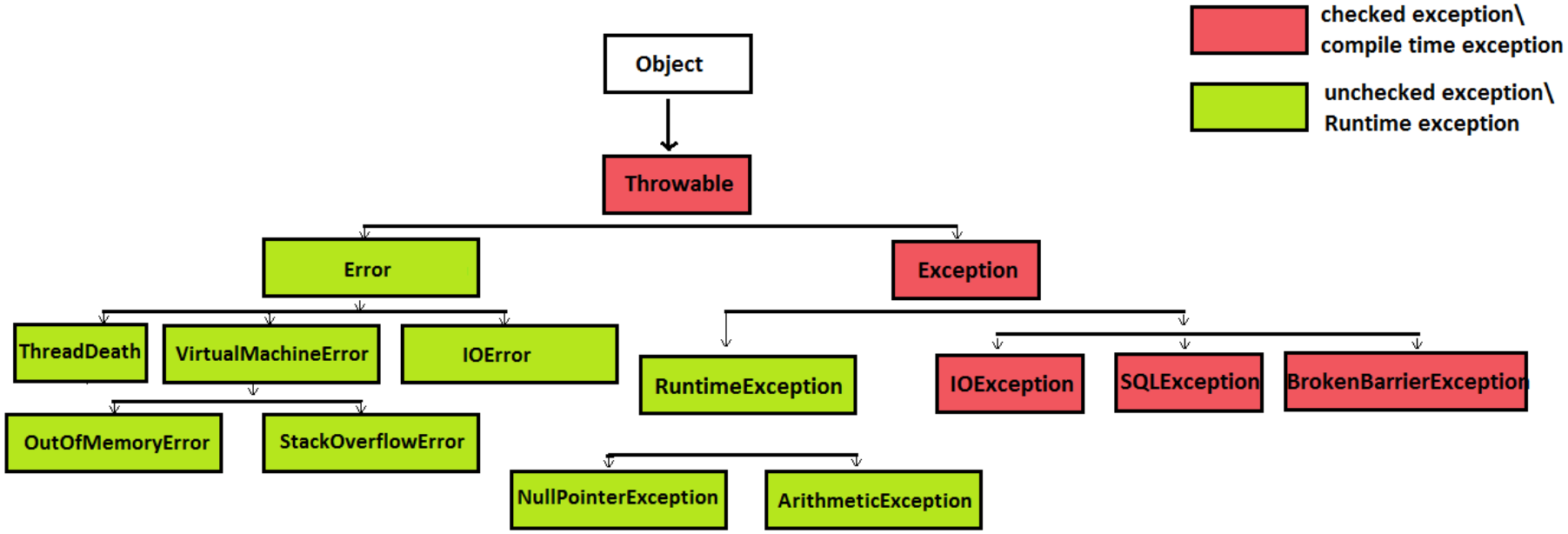
# Исключения

# Error и Exception

- В чем разница?
- Виды исключений?

# Error Exception

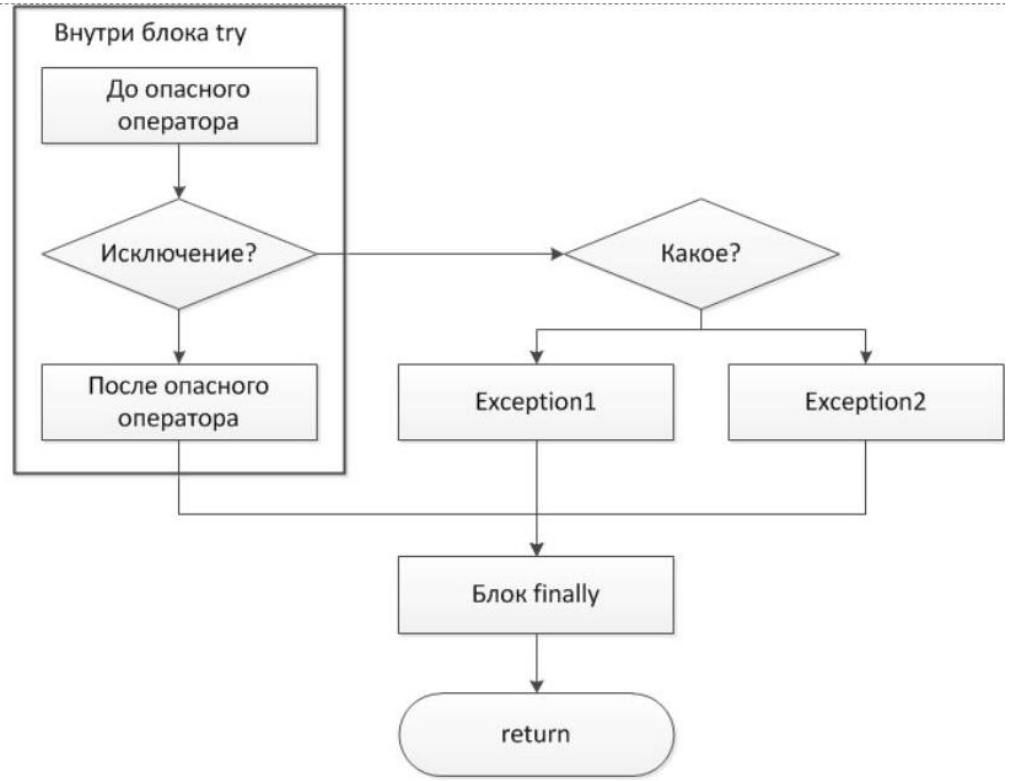



**Это ситуации, которые разработчик никак не может предотвратить, например, не получилось закрыть файловый дескриптор или отослать письмо, и исключение является одним из вариантов нормальной работы кода.**
**Это проверяемые исключения, мы обязаны на такие исключения реагировать, это будет проверено на этапе компиляции.**

**Это ситуации, когда основной причиной ошибки является сам разработчик, например, происходит обращение к null ссылке, деление на ноль, выход за границы массива и т.д. При этом исключение не является одним из вариантов нормальной работы кода.**

**Это критические ошибки, аварийные ситуации, после которых мы с трудом или вообще не в состоянии продолжить работу. Например, закончилась память, переполнился стек вызовов и т.д.**

# Обработка исключений




```
try {
   // здесь возможно возникновение исключения
} catch(тип_исключения1 переменная1) {
   // обработчик исключения типа тип_исключения1
} catch(тип_исключения2 переменная2) {
   // обработчик исключения типа тип_исключения2
}
// ...
finally {
   // код, который выполняется в любом случае
после выполнения блока try или завершения
обработки исключения в блоке catch
```

# Обработка исключений

Блок `try` находится перед блоком `catch` или `finally`. При этом должен присутствовать хотя бы один из этих блоков.

Между `try`, `catch` и `finally` не может быть никаких операторов.

Один блок `try` может иметь несколько `catch` блоков. В таком случае будет выполняться первый подходящий блок.
Поэтому сначала должны идти более специальные блоки обработки исключений, а потом уже более общие.

Блок `finally` будет выполнен всегда, кроме случая,
когда `JVM` преждевременно завершит работу или будет сгенерировано исключение непосредственно в самом `finally` блоке.

- **Почему лучше не "ловить" Throwable t?**
- **Для чего нужен блок finally**

# "Пробрасывание" исключений

Для генерации исключений в Java предназначен оператор **throw**, которому передаётся объект исключения. Обычно этот объект создаётся непосредственно при вызове оператора **throw**.

Пример:

ArithmeticException e = **new** ArithmeticException();

**throw** e;

или

**throw new** Exception();

```
//: exceptions/InheritingExceptions.java
// Создание собственного исключения.
import com.bruceeckel.simpletest.*;

class SimpleException extends Exception {}

public class InheritingExceptions {
  public void f() throws SimpleException {
    System.out.println("Возбуждаем SimpleException из f()");
    throw new SimpleException();
  }
  public static void main(String[] args) {
    InheritingExceptions sed = new InheritingExceptions();
    try {
      sed.f();
    } catch(SimpleException e) {
      System.out.println("Перехвачено!");
    }
  }
} /* Output:
Возбуждаем SimpleException из f()
Перехвачено!
*///:~
```

Когда мы можем пробросить исключение?

# Прерывание и возобновление

- Прерывание - при возникновение ошибки, система дает понять о ее наличии, не предпринимая действий по ее устранению
- Возобновление - обработчик ошибок сделает что-то для исправления ситуации и запускает код заново

- Как осуществить модель возобновления на Java?
- Почему концепцию прерывания используют чаще?

# Оператор throws

**Если внутри функции может быть сгенерировано исключение, необработанное с помощью конструкции try, после объявления этой функции должно стоять зарезервированное слово throws и тип генерируемого исключения.**

**Пример:**

```
public void someFunction() throws SomeException {
// ...
throw new SomeException();
// ...
}
```

- Информация о исключениях
- Создание отдельного логировщика для исключений для более полного вывода информации
- Спецификация исключений- обязательный синтекст для сообщения том, что метод возбуждает исключения
- Следует сразу за списком аргументов
- someFunc(someargs) throw someExeption{} - либо так либо обработка исключений
- можно возбудить исключение которого нет, используется для абстрактных классов и интерфейсов
- Перехват любого типа исключений -> лучше в конец списка,тк поочереность и не будут
- перехвачены другие
- Трассировка стэка
- printStackTrace - озвращает массив элементов трассировки, каждый элемент представляет один кадр
- стека
- Нулевой элемент - на вершине стека
- Повторное возбуждение исключений
- ТК ссылка на исключение уже есть, надо просто пробросить еще раз -> переход в распоряжение
- обраюотчика более высокого уровня, все блоки кэтч игнорируются
- Если просто заново возбудить исключение, то информация о месте первого вызова исключения
- для обновленя информации - fillInStackTrace()
- есть возможность повторного возбуждения,отличающееся от перехваченного - вызов сначало во
- внутреннем блоке try, затем во внешнем

- Цепочки исключений
- перехват одного и возбуждение следующих исключений
- подклассы trowable могут принимать в качетве аргумента конструктора объект-причину
- Цепочка исключений как конструктор может быть только в error, exeption, runtimeexeption
- Для других нужно использовать - initCause

- Стандартные исключения Java
- Все, что может быть искл - Throwable
- Основное отличие исключений - имя

- RuntimeException
- Возбуждаются автоматически, нет нужды в их спецификации
- Перехватываются автоматически
- Могут быть проигнорированы в программном коде, обработка остыльных - засчет компилятора
- может произойти из-за непредвиденной ошибки\ошибка, которую надо было не делать

- Потерянное исключение
- происходит при исп finally - return; - подавляет любое искл

- Ограничение исключений
- при переопр метода вправе возбуждать только те искл, которые описаны в методе базового класса
- (или интерфейсе)
- конструктор унаследованного класса не может перехватывать исключения, вызываемые конструкторо базы

- Исключение в конструкторе
- опасно использовать finally тк исключение может пройзойти, а объект не до конца построен
- Самый безопасный метод - вложенные блоки try, finally - связанный со внутренним блоком и выполняетс не всегда

- Отождествление исключений - не требует точного соответсвия между исключением и обработчиком
- важно - не обраб искл, пока неизвестно что с ним делать
- Преобразование контролируемых исключений в неконтролируемые
- обертка исключений в рантайм исключение
- проброс собственного

- Исключения для
- Обработки ошибки
- Исправление ошибки и повторный вызов метода
- Исправление ошибки и не вызывать метод
- Альтернативный результат
- Сделать все что можно и возбудить это же(новое) исключение на более высоком уровне
- завершение работы программы
- упростить программу
- повысть уровень безопасности библиотеки