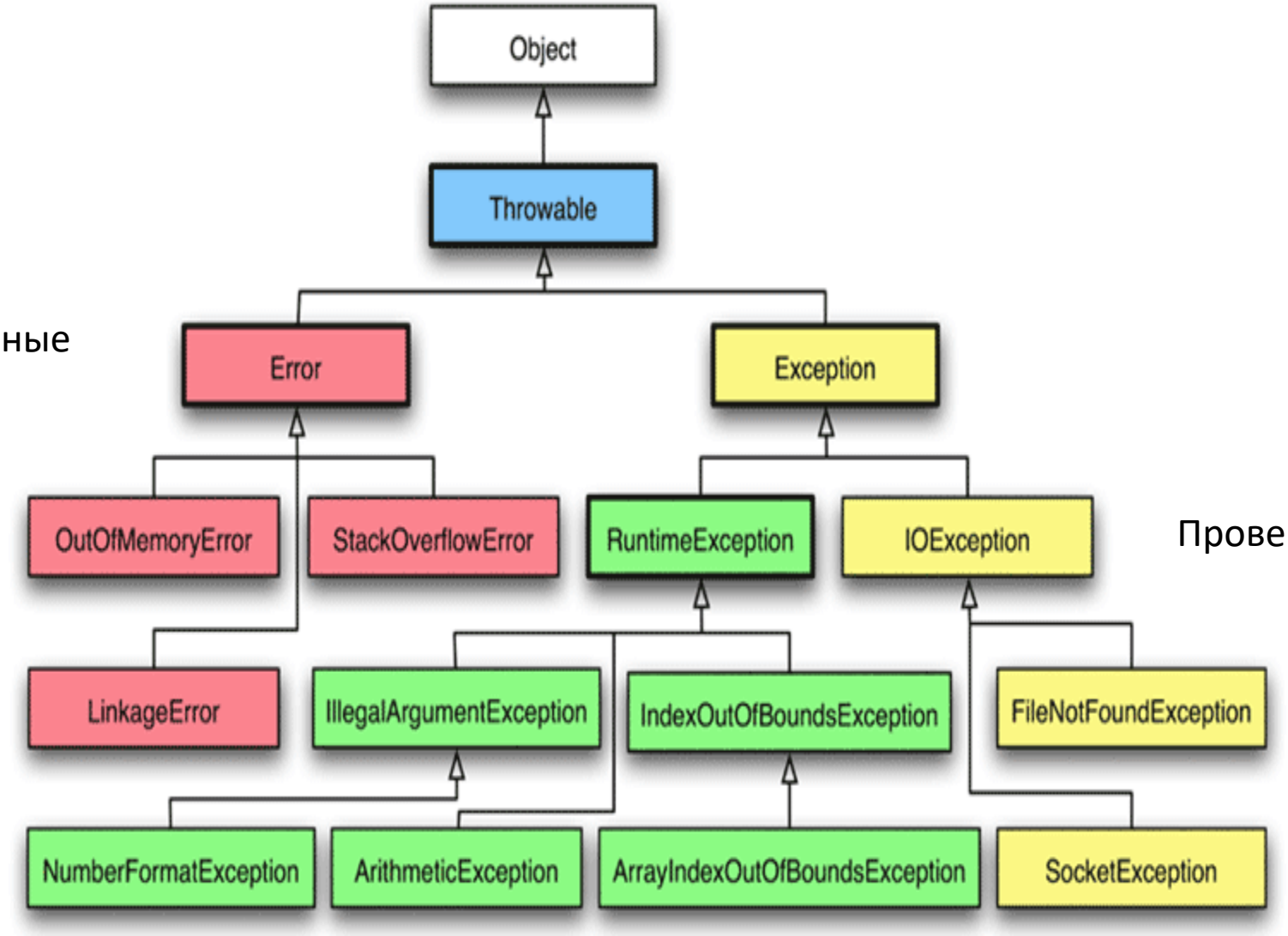
Исключения (Exceptions)

Исключениями или исключительными ситуациями (состояниями) называются события, возникшие в программе во время её работы.

Причины возникновения исключений:

- Попытка выполнить некорректное выражение. Например, деление на ноль, или обращение к объекту по ссылке, равной null, попытка использовать класс, описание которого отсутствует, и т.д.
- Выполнение оператора throw. Этот оператор применяется для явного порождения исключения.
- Асинхронные ошибки во время исполнения программы. Причиной таких ошибок могут быть сбои внутри самой виртуальной машины

Иерархия классов стандартных исключений



Фатальные

Проверяемые

Обрабатываемые

Проверяемые исключения

Ошибки, порожденные от Exception и не являющиеся наследниками RuntimeException (желтый цвет), являются проверяемыми. Возникают при вызове определенных методов, т.е. во время компиляции проверяется, предусмотрена ли обработка возможных исключительных ситуаций. Как правило, это ошибки, связанные с окружением программы (сетевым, файловым вводом-выводом и др.), которые могут возникнуть вне зависимости от того, корректно написан код или нет.

Метод обязан отреагировать на исключение, либо обработать его, либо передать его вызывающему методу.

Таким образом повышается надежность программы, ее устойчивость при возможных сбоях.

Пример проверяемых исключений

```
1. import java.io.File;
2. import java.io.FileReader;
3. public class FilenotFound_Demo {
4.     public static void main(String args[]) {
5.         File file = new File("E://file.txt");
6.         FileReader fr = new FileReader(file);
7.     }
8. }
```

Если файл, указанный в конструкторе FileReader, не существует, то возникнет исключение FileNotFoundException, поэтому компилятор предлагает программисту обработать исключение.

```
FilenotFound_Demo.java:6: error: unreported exception
FileNotFoundException; must be caught or declared to be thrown
FileReader fr = new FileReader(file);
                ^
1 error
```

Обрабатываемые непроверяемые исключения

Исключения, порожденные от RuntimeException (зеленый цвет), являются непроверяемыми и компилятор не требует обязательной их обработки. Как правило, это ошибки программы, которые при правильном кодировании возникать не должны. Поэтому, чтобы не загромождать программу, компилятор оставляет на усмотрение программиста обработку таких исключений с помощью блоков try-catch.

- ArithmeticException - ошибки вычислений.
- NullPointerException - ссылка на пустое место.
- NegativeArraySizeException - массив отрицательной размерности.
- ArrayStoreException - присвоение элементу массива неправильного типа.
- NumberFormatException - невозможно преобразовать строку в число.
- IllegalArgumentException - неправильный аргумент при вызове метода.
- UnsupportedOperationException - указанной операции не существует.
- TypeNotPresentException - указанного типа не существует.

Фатальные непроверяемые исключения

Исключения, порожденные от Error (красный цвет). Они предназначены для того, чтобы уведомить приложение о возникновении фатальной ситуации, которую программным способом устранить практически невозможно (хотя формально обработчик допускается). Они могут свидетельствовать об ошибках программы, но, как правило, это неустранимые проблемы на уровне JVM. В качестве примера можно привести StackOverflowError (переполнение стека), OutOfMemoryError (нехватка памяти).

Основные методы базового типа Throwable

Throwable fillInStackTrace() - возвращает объект класса **Throwable**, содержащий полную трассировку стека.

String getMessage() - возвращает подробное описание исключения.

StackTraceElement[] getStackTrace() - обеспечивает программный доступ к информации трассировки стека.

void printStackTrace() - отображает трассировку стека.

void printStackTrace(PrintStream stream) – посылает. трассировку стека в заданный поток (на консоль или в файл).

void printStackTrace(PrintWriter stream) - посылает трассировку стека в заданный поток.

String toString() - возвращает краткое описание исключения.

Неперехваченные исключения

```
1. package etu.lab.exp;
2. classExc 0 {
3. public static void main(string args[]) {
4. int d = 0;
5. int a = 42 / d;
6. }
7. }
```

Exception in thread "main" java.lang.ArithmeticException: / by zero

at etu.lab.exp.Exc0.main(Exc0.java:5)

```
1. package etu.lab.exp;
2. class Exc0 {
3. static void subroutine() {
4. int d = 0;
5. int a= 10 / d;
6. }
7. public static void main(String args[]) {
8. Exc0 .subroutine();
9. }
10. }
```

Exception in thread "main" java.lang.ArithmeticException: / by zero

at etu.lab.exp.Exc0.subroutine(Exc0.java:5) at etu.lab.exp.Exc0.main(Exc0.java:8)

Использование оператора throw и throws

Оператор throw используется для возбуждения исключения «вручную».

```
public int calculate(int theValue)
{ if( theValue < 0) { throw new Exception( "Параметр для вычисления не должен быть отрицательным");
// Обработка исключительной ситуации
} }
```

Если метод способен возбуждать исключения, которые он сам не обрабатывает, он должен объявить о таком поведении, чтобы вызывающие методы могли защитить себя от этих исключений.

```
public int calculate(int theValue) throws Exception
{ if( theValue < 0) { throw new Exception( " Параметр для вычисления не должен быть отрицательным "); } }
```

Без обработки исключение, где вызывается функция, возникнет ошибка компиляции.

Обработка исключений

try { // Код, который может сгенерировать исключение }

catch(Type1 id1) { // Обработка исключения Type1 }

catch(Type2 id2) { // Обработка исключения Type2 }

catch(Type3 id3) { // Обработка исключения Type3 }

// продолжение программы после обработки исключения

Обработчикам не доступны переменные, объявленные в блоке try

Обработчики подклассов исключений должные находиться выше, чем обработчики их суперклассов, в противном случае будет ошибка компиляции.

try { ... }

catch(IOException ex) {// Перехват исключений, когда операция ввода-вывода // завершилась неудачно или прервана.

// Обработка исключительной ситуации

...

// Повторное возбуждение исключительной ситуации. Передача исключения обработчику верхнего уровня

throw ex; }

catch(Exception e) { // Перехват программных исключений }

catch(Throwable t) { // Перехват любого исключения }

Конструкция try-catch-finally

```
try { // Критическая область, при которой могут быть выброшены исключения A, B или C }
catch(A a1) { // Обработчик ситуации A }
catch(B b1) { // Обработчик ситуации B }
catch(C c1) { // Обработчик ситуации C }
finally { // Действия, совершаемые всякий раз }
int getNumber(){
 try { //Исключения есть или нет
return 0;
}
catch(Exception e){
return 1;
 }
 finally {
return 2;
}
return 3;
}
```

Ответ: 2.

Обработка нескольких исключений в одном блоке catch (Java 7)

```
try {
 ...
} catch( IOException | SQLException ex ) {
  System.out.println(ex.getMessage());// вывод информации о исключении на консоль
  throw ex;  // Повторное возбуждение исключительной  ситуации
}
```

Каждый оператор try требует наличия либо catch, либо finally, либо сочетания catch и finally.

Если при исключении блок catch не найден, **JVM** останавливает выполнение программы, и выводит стек вызовов методов – stack trace, выполнив перед этим код блока finally при его наличии.

Пример обработки исключения

```
class ThrowDemo {
    static void demoproc() {
    {
            throw new NullPointerException("demo");
    }
    catch (NullPointerException e) {
            System.out.println(“обработка внутри demoproc");
             throw e;  // повторный вызов исключения
    }
    }
    public static void main(String args[]) {
    try {
              demoproc();
    }
    catch(NulPointerException e) {
               System.out.println(“повторно:" + e);
    }
    }
    }
```

Вложенные операторы try

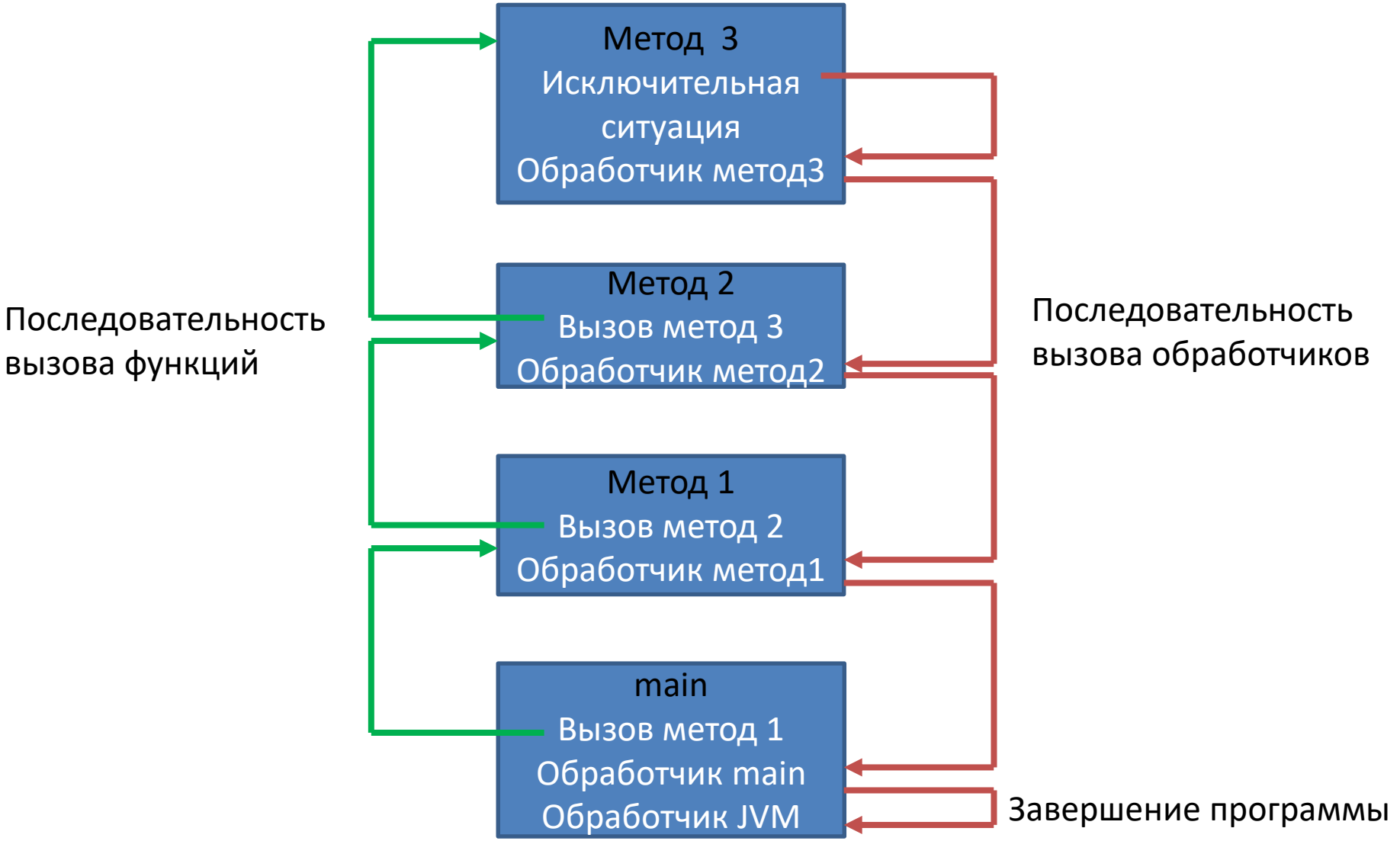
```
class NestTry {
public static void main(String args[]) {
try {
        int a = args.length;
        /* Если не указаны параметры командной строки, следующий оператор сгенерирует исключение
             деления на ноль. *'/
        int b = 42 / a;
        System.out.println("a = " + a) ;
   try { // вложенный блок try
/* Если используется один аргумент командной строки, то исключение деления на ноль будет
      сгенерировано следующим кодом. */
     if(a==l) a = a/(a-a); // деление на ноль
/* Если используется два аргумента командной строки, то генерируется исключение выхода за пределы
      массива. */
        if(a==2) {
        int c[] = { 1 };
        c [42] = 99; // генерируется исключение выхода за пределы массива
                  }
       }
        catch(ArrayIndexOutOfBoundsException e) {
        System.out.println("Индекс за пределами массива: " + e) ;
        }
}
catch(ArithmeticException e) {
System.out.println("Деление на 0: " + e);
}
}
}
```

```
Деление на 0: java.lang.ArithmeticException: / by zero

a = 1
Деление на 0: java.lang.ArithmeticException: / by zero

a = 2
Индекс за пределами массива:
j ava.lang.ArraylndexOutOfBoundsException:42
```

Стек вызовов



Создание пользовательских классов исключений

```
class FactorialException extends Exception{ // исключения для проверки вычисления факториала на < 1
    private int number;
   public int getNumber(){return number;}
   public FactorialException(String message, int num){  // конструктор
          super(message); // в базовый класс передаем сообщение
      number=num;
   }
}
public class Factorial{
    public static int getFactorial(int num) throws FactorialException{
          int result=1;
      if(num<1) throw new FactorialException("Число не может быть меньше 1", num);
            for(int i=1; i<=num;i++){ result*=i;
      }
      return result;
   }

public static void main(String[] args){
          try{
      int result = Factorial.getFactorial(-6);
      System.out.println(result);
   }
   catch(FactorialException ex){
      System.out.println(ex.getMessage()); // вывод сообщения
      System.out.println(ex.getNumber());
   }
}
 }
```

Переопределение методов и исключения

```
public class BaseClass{
public void method () throws IOException { ... }
 }
public class LegalOne extends BaseClass {
public void method () throws IOException { ... } // корректно (список ошибок не изменился);
 }
public class LegalTwo extends BaseClass {
public void method () { ... }           // корректно (новый метод не может выбрасывать ошибок)
}
public class LegalThree extends BaseClass {
public void method () throws EOFException { ... }  // корректно
// (новый метод может создавать  исключения, которые являются подклассами исключения)
}
public class IllegalOne extends BaseClass {
public void method () throws IOException, IllegalAccessException { ... } // некорректно
// ( IllegalAccessException не является подклассом IOException, список расширился);
}
public class IllegalTwo extends BaseClass {
public void method () { ... throw new Exception(); }         } // некорректно
//(в теле метода бросается исключение, не указанное в throws)
 }
```

Основные правила обработки исключений

- обработать ошибку на текущем уровне (избегайте перехватывать исключения, если не знаете, как с ними поступить)
- исправить проблему и снова вызвать метод, возбудивший исключение
- предпринять все необходимые действия и продолжить выполнение без повторного вызова действия
- сделать все возможное в текущем контексте и заново возбудить это же исключение, перенаправив его на более высокий уровень
- сделать все, что можно в текущем контексте, и возбудить новое исключение, перенаправив его на более высокий уровень
- поскольку исключения останавливают обработку программы, необходимо закрыть все ресурсы в блоке finally
- завершить работу программы