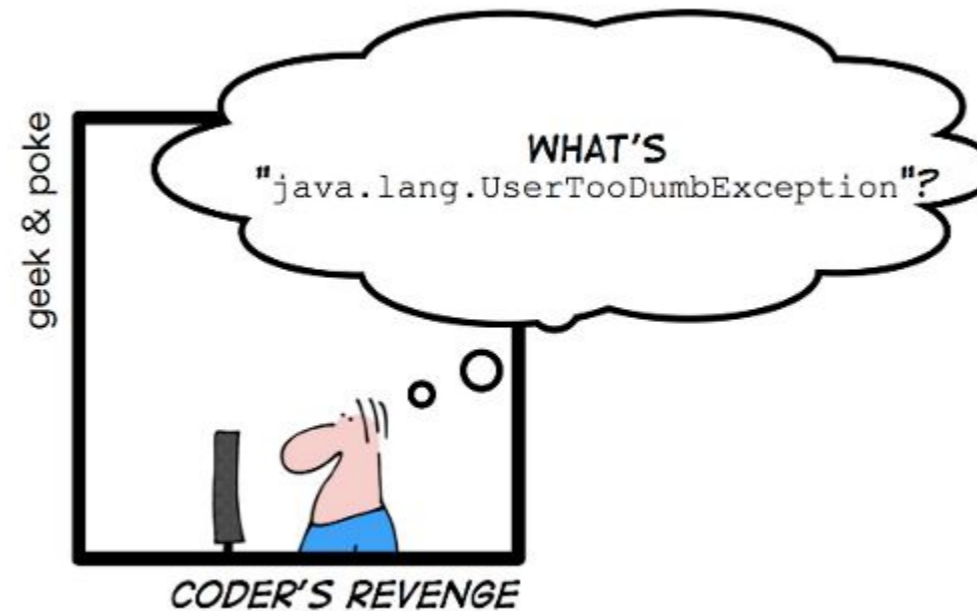






VI. Исключения

1. Использование исключений



Исключительная ситуация - ошибка времени выполнения из-за которой нормальное продолжение работы выполняемого метода становится невозможным. Когда возникает исключительная ситуация среда выполнения (runtime environment) или код обнаруживший ошибку "выбрасывает" исключение. Выбрасывание исключения даёт возможность изменить путь выполнения программы когда происходит исключительная ситуация.



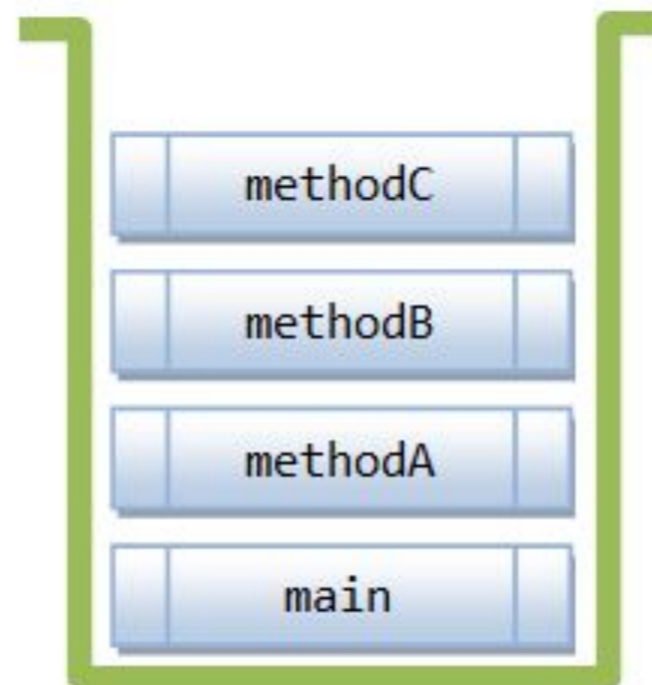
Исключение - объект описывающий исключительную ситуацию. Исключения могут создаваться и выбрасываться средой выполнения (runtime environment) или программой. Исключение как правило содержит достаточно информации чтобы указать на то где ошибка произошла, какая ошибка произошла и данные которые привели к ошибке или которые указывают на ошибку.

```
public class NoCatchDemo {  
  
    public static void main(String[] args) {  
        System.out.println("Enter main()");  
        int result = 1 / 0;  
        System.out.println("Exit main()");  
    }  
  
}
```

Enter main()

Exception in thread "main" java.lang.ArithmeticException: / by zero
at usage.NoCatchDemo.main(NoCatchDemo.java:7)

Стек вызовов



**Method Call Stack
(Last-in-First-out Queue)**

```
public class CallStackDemo {
    public static void main(String[] args) {
        System.out.println("Enter main()");
        methodA();
        System.out.println("Exit main()");
    }

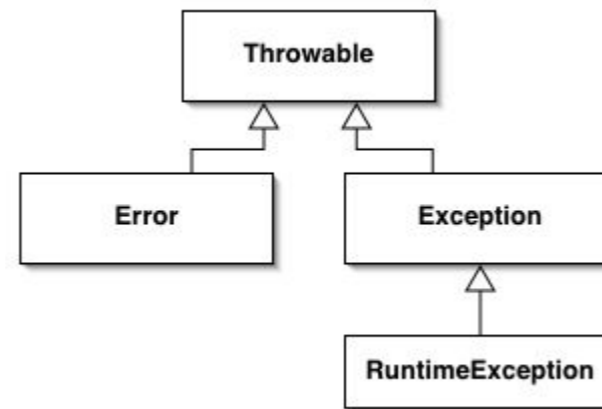
    public static void methodA() {
        System.out.println("Enter methodA()");
        methodB();
        System.out.println("Exit methodA()");
    }

    public static void methodB() {
        System.out.println("Enter methodB()");
        methodC();
        System.out.println("Exit methodB()");
    }

    public static void methodC() {
        System.out.println("Enter methodC()");
        int result = 1 / 0;
        System.out.println("Exit methodC()");
    }
}
```

```
Enter main()
Enter methodA()
Enter methodB()
Enter methodC()
Exception in thread "main" java.lang.ArithmeticException: / by zero
at usage.CallStackDemo.methodC(CallStackDemo.java:28)
at usage.CallStackDemo.methodB(CallStackDemo.java:22)
at usage.CallStackDemo.methodA(CallStackDemo.java:16)
at usage.CallStackDemo.main(CallStackDemo.java:10)
```

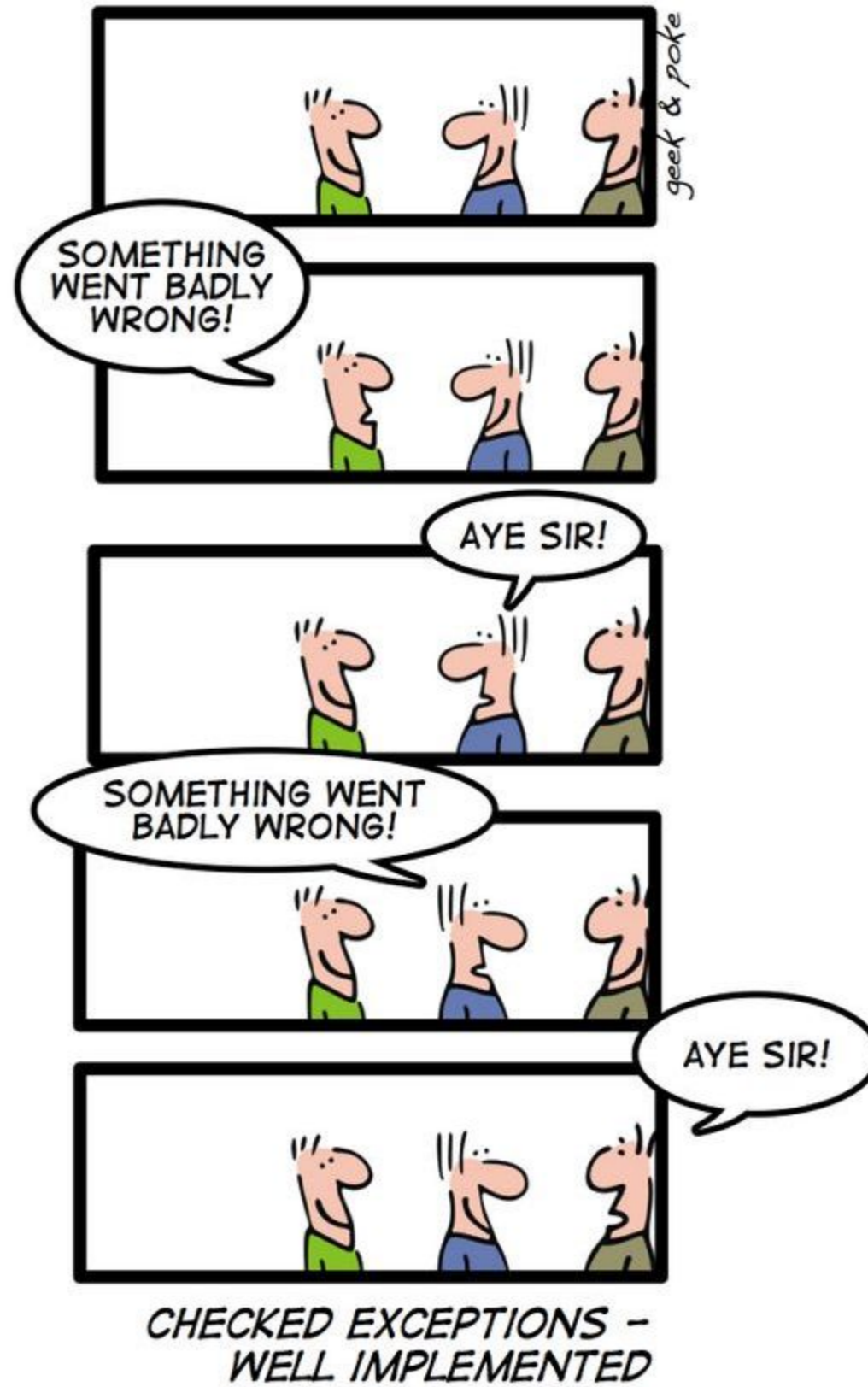
Контролируемые и неконтролируемые исключения

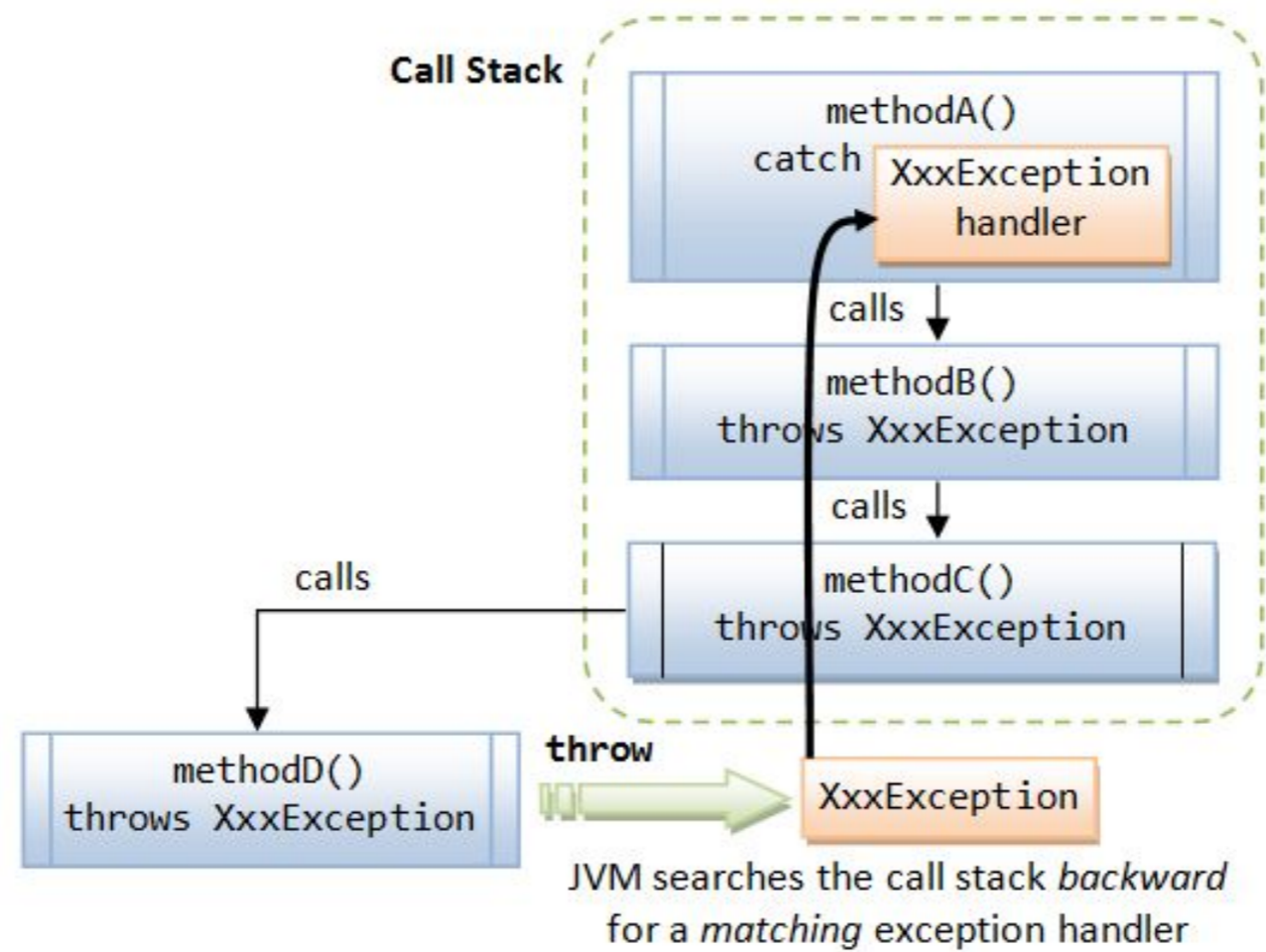


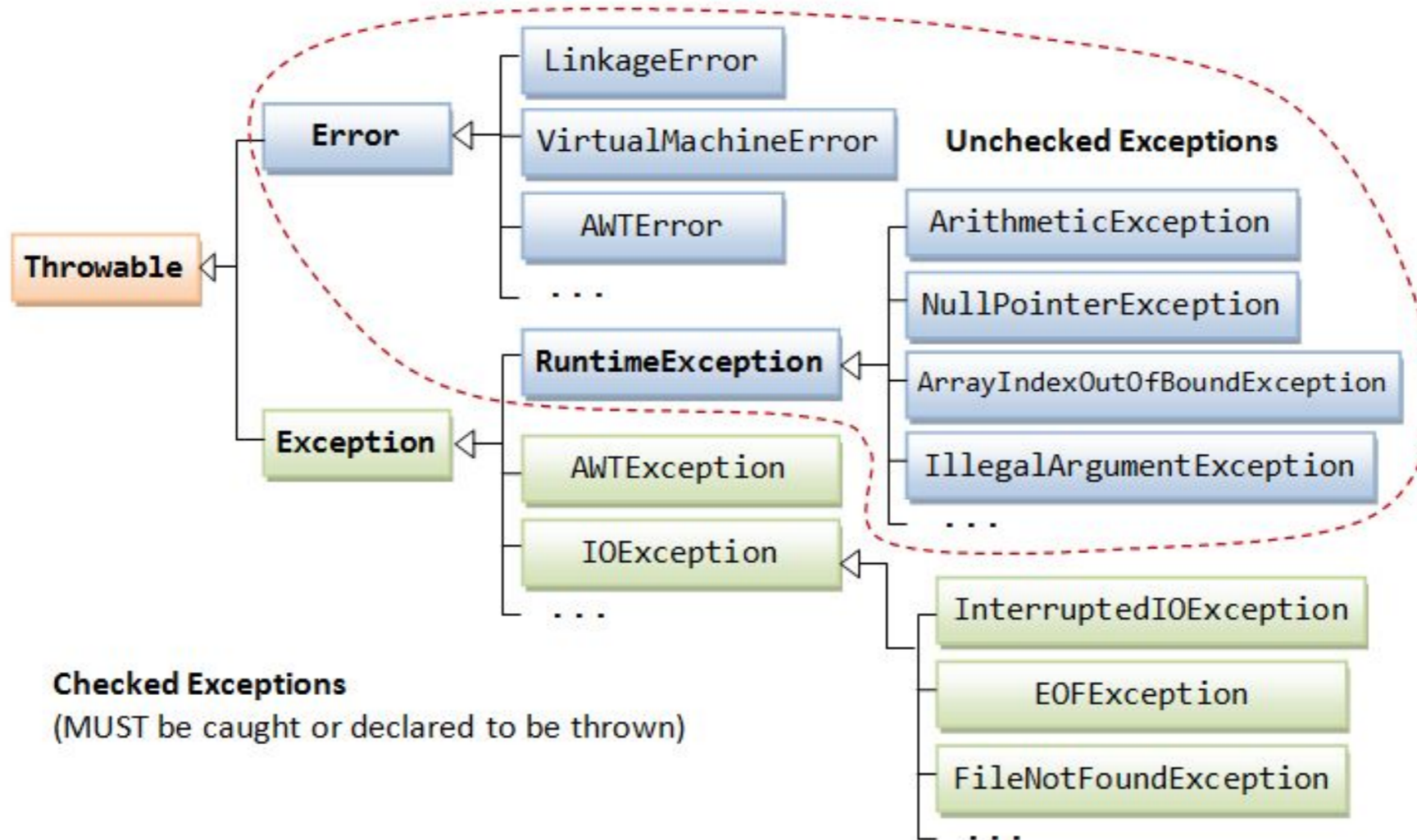
Исключения делятся на контролируемые и неконтролируемые. Классы неконтролируемых исключений являются потомками класса RuntimeException или класса Error. Классы контролируемых исключений являются потомками класса Exception, но не являются потомками класса RuntimeException. Контролируемые исключения которые метод может выбрасывать должны указываться с помощью ключевого слова throws при объявлении метода. Компилятор **контролирует** что контролируемые исключения которые могут быть выброшены в методе перехватываются или объявляются с помощью ключевого слова throws.



Неконтролируемые исключения включают ошибки - классы потомки класса Error. Исключения типа Error используются средой выполнения Java для обозначения ошибок происходящих внутри самой среды. Обычно они создаются в ответ на катастрофические сбои после которых программа **не может продолжить выполнение**.







Почему контролируемые исключения?



Выбрасывание неконтролируемых исключений (за исключением потомков класса `Error`) можно предотвратить и они означают ошибку разработчика. Контролируемые исключения связаны с состоянием среды в которой программа выполняется и их нельзя предотвратить. Таким образом компилятор заставляет программу быть готовой к непредотвратимым ситуациям связанным с состоянием среды и программа надёжна.

```
public class UncheckedDemo {  
  
    public static void main(String[] args) {  
  
        Scanner input = new Scanner(System.in);  
  
        System.out.println("Enter first number: ");  
        int n1 = input.nextInt();  
  
        System.out.println("Enter second number: ");  
        int n2 = input.nextInt();  
  
        int result = n1 / n2;  
        System.out.println("The result is: " + result);  
    }  
}
```

Enter first number:

10

Enter second number:

abcdef

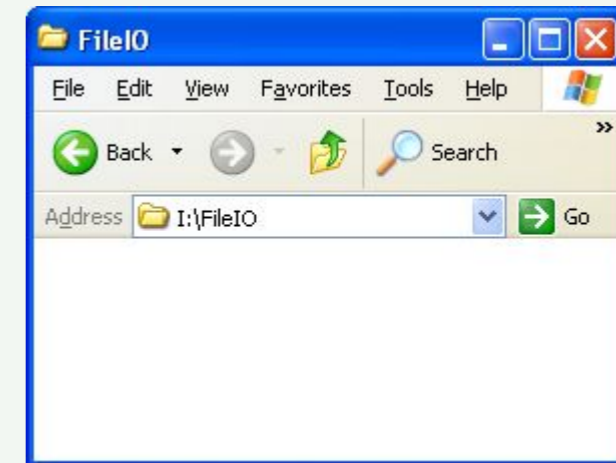
```
Exception in thread "main" java.util.InputMismatchException  
at java.util.Scanner.throwFor(Unknown Source)  
at java.util.Scanner.next(Unknown Source)  
at java.util.Scanner.nextInt(Unknown Source)  
at java.util.Scanner.nextInt(Unknown Source)  
at usage.UncheckedNoCatchDemo.main(UncheckedNoCatchDemo.java:15)
```

```
public class UncheckedPreventDemo {  
  
    public static void main(String[] args) {  
  
        Scanner input = new Scanner(System.in);  
  
        System.out.println("Enter first number: ");  
        while (!input.hasNextInt()) {  
            input.next();  
            System.out.println("Wrong format ..... \nEnter first number: ");  
        }  
        int n1 = input.nextInt();  
  
        System.out.println("Enter non-zero second number: ");  
  
        int n2 = 0;  
        do {  
            while (!input.hasNextInt()) {  
                input.next();  
                System.out.println("Wrong format ..... \nEnter second number: ");  
            }  
            n2 = input.nextInt();  
            if (n2 == 0)  
                System.out.println("Second number can not be zero ..... \nEnter second number: ");  
        } while (n2 == 0);  
  
        int result = n1 / n2;  
        System.out.println("The result is: " + result);  
    }  
}
```



```
Enter first number:  
abcde  
Wrong format .....  
Enter first number:  
125  
Enter non-zero second number:  
fghij  
Wrong format .....  
Enter non-zero second number:  
0  
Second number can not be zero .....  
Enter non-zero second number:  
5  
The result is: 25
```

```
public class CheckedNoCatchDemo {  
  
    public static void main(String[] args) throws FileNotFoundException {  
  
        Scanner consoleIn = new Scanner(System.in);  
        System.out.println("Enter file name: ");  
        String fileName = consoleIn.nextLine();  
  
        Scanner fileIn = new Scanner(new File(fileName));  
  
        int count = 0;  
        while (fileIn.hasNext()) {  
            String word = fileIn.next();  
            count++;  
        }  
        System.out.println("Number of words is " + count);  
        fileIn.close();  
    }  
}
```



Enter file name:

I:\FileIO\hello.txt

Exception in thread "main" java.io.FileNotFoundException: I:\FileIO\hello.txt (The system cannot find the file specified)

at java.io.FileInputStream.open(Native Method)

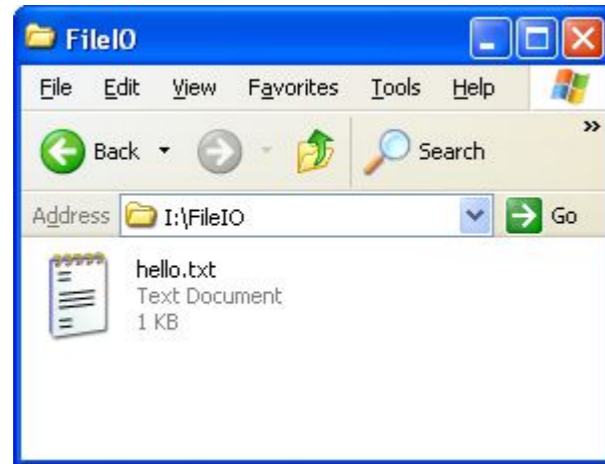
at java.io.FileInputStream.<init>(Unknown Source)

at java.util.Scanner.<init>(Unknown Source)

at usage.CheckedNoCatchDemo.main(CheckedNoCatchDemo.java:16)

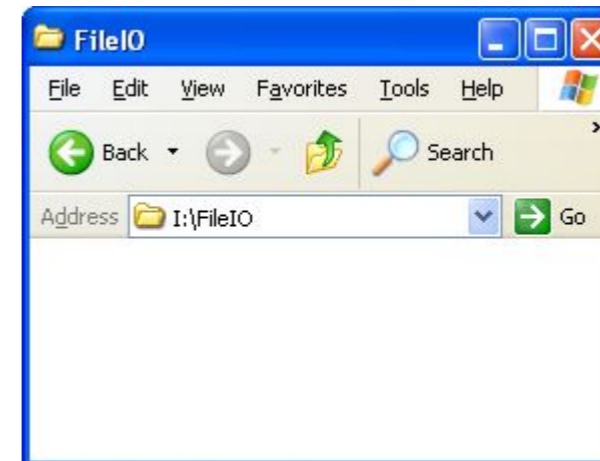
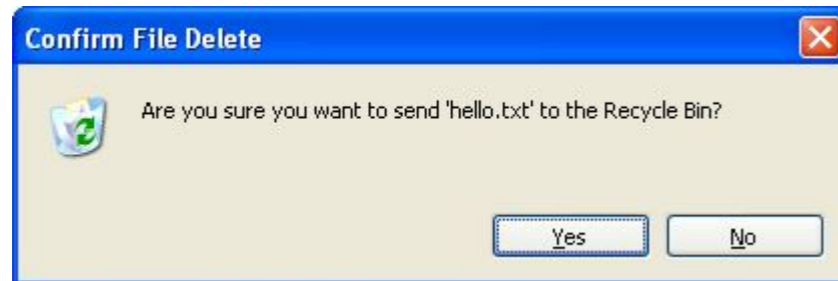
```
public class CheckedDemo {  
  
    public static void main(String[] args) throws FileNotFoundException {  
  
        Scanner consoleIn = new Scanner(System.in);  
        File file = null;  
  
        do {  
            System.out.println("Enter file name: ");  
            String fileName = consoleIn.nextLine();  
            file = new File(fileName);  
  
        } while (!file.exists());  
  
        try {  
            Thread.sleep(10000);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
  
        Scanner fileIn = new Scanner(file);  
  
        int count = 0;  
        while (fileIn.hasNext()) {  
            String word = fileIn.next();  
            count++;  
        }  
        System.out.println("Number of words is " + count);  
        fileIn.close();  
    }  
}
```

Попытка предотвратить контролируемое исключение



Enter file name:

I:\FileIO\hello.txt



```
Exception in thread "main" java.io.FileNotFoundException: I:\FileIO\hello.txt (The system cannot find the file specified)
at java.io.FileInputStream.open(Native Method)
at java.io.FileInputStream.<init>(Unknown Source)
at java.util.Scanner.<init>(Unknown Source)
at usage.CheckedDemo.main(CheckedDemo.java:27)
```

Перехватывание исключений



Если исключение не перехватывается программой оно перехватывается стандартным обработчиком исключений. Стандартный обработчик выводит тип исключения, строку описывающую исключение и трассировку стека. Для перехватывания и обработки исключения можно поместить фрагмент кода который может выбросить исключение в блок `try`, а код для обработки исключения в блок `catch`. В операторе `catch` указывается тип перехватываемых исключений. Когда выбрасывается исключение выполнение программы останавливается и выполнение передаётся блоку `catch` перехватывающему исключение.

```
try {  
    statement_sequence  
} catch (ExceptionType1 id1) {  
    another_statement_sequence  
}
```

```
public class CheckedCatchDemo {  
  
    public static void main(String[] args) {  
  
        Scanner fileIn = null;  
  
        do {  
            Scanner consoleIn = new Scanner(System.in);  
            System.out.println("Enter a file name: ");  
            String fileName = consoleIn.nextLine();  
            try {  
                fileIn = new Scanner(new File(fileName));  
            } catch (FileNotFoundException e) {  
                System.out.println("File not found");  
            }  
        } while (fileIn == null);  
  
        int count = 0;  
        while (fileIn.hasNext()) {  
            String word = fileIn.next();  
            count++;  
        }  
        System.out.println("Number of words is " + count);  
        fileIn.close();  
    }  
}
```

```
Enter a file name:  
hello world!  
File not found  
Enter a file name:  
I:\FileIO\words.txt  
Number of words is 16
```


Множественные операторы catch



Если фрагмент кода может выбрасывать более одного исключения можно использовать несколько операторов catch каждый для перехвата своего типа исключений. Когда выбрасывается исключение операторы catch проверяются по порядку и первый тип которого позволяет обработать исключение выполняется. После того как выполнен один оператор catch остальные операторы пропускаются и выполнение продолжается после блока try/catch. Оператор catch для класса исключения потомка должен идти до оператора catch для класса исключения предка в противном случае будет ошибка компиляции.

```
try {  
    statement_sequence  
} catch (ExceptionType1 id1) {  
    statement_sequence1  
} catch (ExceptionType2 id2) {  
    statement_sequence2  
} catch (ExceptionType3 id3) {  
    ...  
} catch (ExceptionTypeN idN) {  
    statement_sequenceN  
}
```

```
public class CheckedMultipleCatchDemo {  
  
    public static void main(String[] args) {  
  
        BufferedReader console = new BufferedReader(new InputStreamReader(System.in));  
  
        while (true) {  
  
            try {  
  
                System.out.println("Please enter the file name: ");  
                String filename = console.readLine();  
  
                FileWriter out = new FileWriter(new File(filename));  
                out.write("Hello World!");  
                out.close();  
  
            } catch (FileNotFoundException e) {  
                System.out.println("File not found");  
            }  
            catch (EOFException e) {  
                System.out.println("End of file reached");  
            }  
            catch (IOException e) {  
                System.out.println("General I/O exception");  
            }  
        }  
    }  
}
```

```
Please enter the file name:  
I:\noSuchDir\noSuchfile.txt  
File not found  
Please enter the file name:
```

Завершение с помощью finally



Блок finally содержит код который будет обязательно выполнен после завершения выполнения блоков try/catch независимо от того было выброшено исключение в блоке try или нет. Как правило он используется для гарантированного освобождения ресурсов. Блок finally указывать не обязательно, но у каждого блока try должен быть по крайней мере один ассоциированный блок catch или блок finally.



В JDK 7 добавлена новая форма оператора try обеспечивающая автоматическое освобождение ресурсов - try-with-resource. Эта форма оператора try защищает исключение выброшенное из блока try от замены исключениями выброшенными при освобождении ресурсов, но позволяет при необходимости их получить.

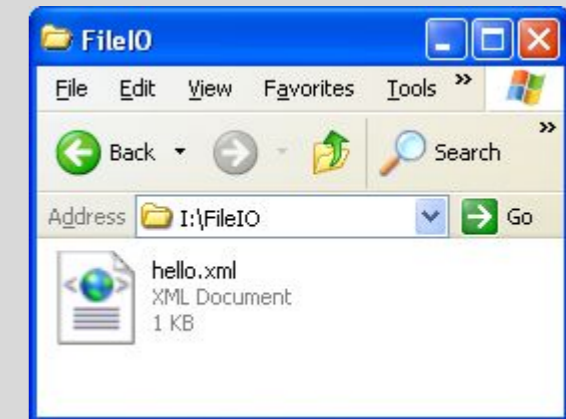
Завершение с помощью finally

```
try {  
    statement_sequence  
} catch (ExceptionType1 id1) {  
    statement_sequence1  
} catch (ExceptionType2 id2) {  
    ...  
} catch (ExceptionTypeN idN) {  
    statement_sequenceN  
} finally {  
    statement_sequence  
}
```



```
public class SimpleFinallyDemo {  
  
    public static void main(String[] args) {  
  
        String name = "I:\\FileIO\\helloWorld.xml";  
  
        BufferedReader reader = null;  
  
        try {  
            reader = new BufferedReader(new FileReader(name));  
            String line = null;  
            while ((line = reader.readLine()) != null) {  
                System.out.println(line);  
            }  
  
        } catch (IOException e) {  
            System.out.println("Problem occurred : " + e.getMessage());  
        } finally {  
            try {  
                if (reader != null) {  
                    System.out.println("Closing reader...");  
                    reader.close();  
                }  
            } catch (IOException e) {  
                System.out.println("Can not close reader : " + e.getMessage());  
            }  
        }  
    }  
}
```

```
Reader opened.  
<greeting>  
    Hello world!  
</greeting>  
Closing reader...
```



```
Problem occurred : I:\FileIO\hello.xml (The system cannot find the file specified)
```



Завершение с помощью finally

```
try {  
    statement_sequence  
} finally {  
    statement_sequence  
}
```

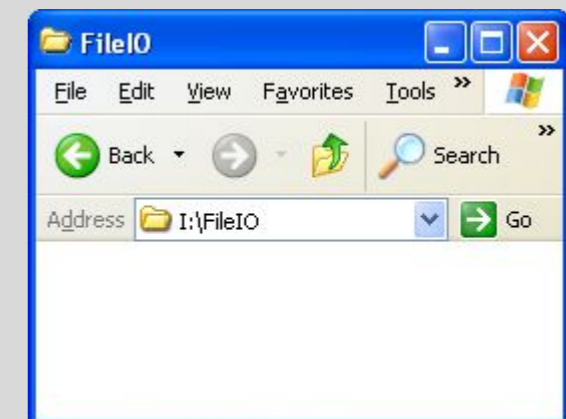
```
public class NestedFinallyDemo {  
  
    public static void main(String[] args) {  
  
        String name = "I:\\FileIO\\hello.xml";  
  
        try {  
            BufferedReader reader = new BufferedReader(new FileReader(name));  
            System.out.println("Reader opened.");  
            try {  
                String line = null;  
                while ((line = reader.readLine()) != null) {  
                    System.out.println(line);  
                }  
            } finally {  
                System.out.println("Closing reader...");  
                reader.close();  
            }  
        } catch (IOException ex) {  
            System.out.println("Problem occurred : " + ex.getMessage());  
        }  
    }  
}
```

Блок finally без catch

```
Reader opened.  
<greeting>  
    Hello world!  
</greeting>  
Closing reader...
```



```
Problem occurred : I:\FileIO\hello.xml (The system cannot find the file specified)
```





Блок finally выполняется всегда даже когда блоки try и catch содержат операторы перехода управления (return, break, continue, throw). Если блок finally включает оператор перехода управления этот оператор заменяет передачу управления из блоков try или catch. По этой причине не следует использовать операторы перехода управления в блоке finally.

```
class FinallyControlDemo {
    static void methodA() {
        try {
            System.out.println("Enter methodA()");
            throw new RuntimeException("demo");
        } finally {
            System.out.println("methodA's finally");
        }
    }

    static void methodB() {
        try {
            System.out.println("Enter methodB");
            return;
        } finally {
            System.out.println("methodB's finally");
        }
    }

    static void methodC() {
        try {
            System.out.println("Enter methodC");
        } finally {
            System.out.println("methodC's finally");
        }
    }

    public static void main(String args[]) {
        try {
            methodA();
        } catch (Exception e) {
            System.out.println("Exception from methodA caught");
        }
        methodB();
        methodC();
    }
}
```

```
Enter methodA()  
methodA's finally  
Exception from methodA caught  
Enter methodB  
methodB's finally  
Enter methodC  
methodC's finally
```



```
class VeryImportantException extends Exception {
    public String toString() {
        return "A very important exception!";
    }
}

class NotSoImportantException extends Exception {
    public String toString() {
        return "Not so important exception";
    }
}

class SomeClass {
    void someMethod() throws VeryImportantException {
        throw new VeryImportantException();
    }
    void close() throws NotSoImportantException {
        throw new NotSoImportantException();
    }
}

public class FinallyReplaceExceptionDemo {
    public static void main(String[] args) {
        try {
            SomeClass some = new SomeClass();
            try {
                some.someMethod();
            } finally {
                some.close();
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Not so important exception

```
public class FinallyLostExceptionDemo {  
  
    public static void main(String[] args) {  
        try {  
            throw new RuntimeException();  
        } finally {  
            System.out.println("Return in finally block will");  
            System.out.println("make any exception disappear");  
  
            return;  
        }  
    }  
}
```

```
Return in finally block will  
make any exception disappear
```

```
class Calculator {  
  
    int someMethod(int i) {  
        try {  
            return i*1000;  
        } finally {  
            return 100;  
        }  
    }  
}  
  
public class FinallyReplaceReturn {  
  
    public static void main(String[] args) {  
  
        Calculator calculator = new Calculator();  
        for (int i = 0; i < 10; i++) {  
            System.out.println(calculator.someMethod(i));  
        }  
    }  
}
```

```
100  
100  
100  
100  
100  
100  
100  
100  
100  
100  
100
```

Конструкторы и блоки инициализации



Конструкторы могут выбрасывать контролируемые и неконтролируемые исключения. Если при вызове конструктора выбрасывается исключение, объект будет создан, но ссылка на него оператором `new` возвращена не будет. Блоки инициализации могут выбрасывать и контролируемые и неконтролируемые исключения. Контролируемые исключения в этом случае необходимо объявлять в каждом конструкторе. Статические блоки инициализации могут выбрасывать только неконтролируемые исключения. Если из статического блока инициализации выбрасывается исключение класс не загружается, а исключение перебрасывается обёрнутое в `ExceptionInInitializerError`.

```
class Person {
    private final String name;
    private final int age;
    private static final int MAXIMUM_AGE = 150;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
        if (this.age < 0 || this.age > MAXIMUM_AGE) {
            throw new IllegalArgumentException("age out of range: " + this.age
                + " expected range 0 <= age < " + MAXIMUM_AGE);
        }
        if (this.name == null) {
            throw new IllegalArgumentException("name is null");
        }
    }

    public String toString() {
        return "Person [name=" + name + ", age=" + age + "]";
    }
}
```

```
public class ConstructorExceptionDemo {  
  
    public static void main(String[] args) {  
  
        Person harry = new Person("Harry Hacker", 25);  
        System.out.println(harry);  
  
        Person tonny = new Person("Tonny Tester", -25);  
        System.out.println(tonny);  
  
    }  
  
}
```

```
Person [name=Harry Hacker, age=25]  
Exception in thread "main" java.lang.IllegalArgumentException: age out of range: -25 expected 0 <= age < 150  
at usage.Person.<init>(ConstructorExceptionDemo.java:27)  
at usage.ConstructorExceptionDemo.main(ConstructorExceptionDemo.java:10)
```

```
class LogManager {  
  
    private static final String FILENAME = "config.txt";  
    private static LogManager manager;  
  
    static {  
        try {  
            InputStream is = new FileInputStream(FILENAME);  
            System.out.println("Reading configuration file");  
            is.close();  
        } catch (IOException ex) {  
            IllegalStateException ise = new IllegalStateException(  
                "Error loading configuration file " + FILENAME);  
            ise.initCause(ex);  
            throw ise;  
        }  
    }  
  
    private LogManager() {  
  
    }  
  
    public static LogManager getLogManager() {  
        if (manager == null)  
            manager = new LogManager();  
        return manager;  
    }  
  
}
```



```
public class StaticInitializerExceptionDemo {  
  
    public static void main(String[] args) {  
        LogManager manager = LogManager.getLogManager();  
    }  
  
}
```

```
Exception in thread "main" java.lang.ExceptionInInitializerError  
at usage.StaticInitializerExceptionDemo.main(StaticInitializerExceptionDemo.java:10)  
Caused by: java.lang.IllegalStateException: Error loading configuration file config.txt  
at usage.LogManager.<clinit>(StaticInitializerExceptionDemo.java:26)  
... 1 more  
Caused by: java.io.FileNotFoundException: config.txt (The system cannot find the file specified)  
at java.io.FileInputStream.open(Native Method)  
at java.io.FileInputStream.<init>(FileInputStream.java:120)  
at java.io.FileInputStream.<init>(FileInputStream.java:79)  
at usage.LogManager.<clinit>(StaticInitializerExceptionDemo.java:22)  
... 1 more
```

Спасибо

**Россия, 127018,
Москва, ул. Полковая 3, стр. 14
Тел.: +7(495) 780 7575, 789 9339
Факс: +7(495) 780 7576, 789 9338
info@diasoft.ru, www.diasoft.ru**