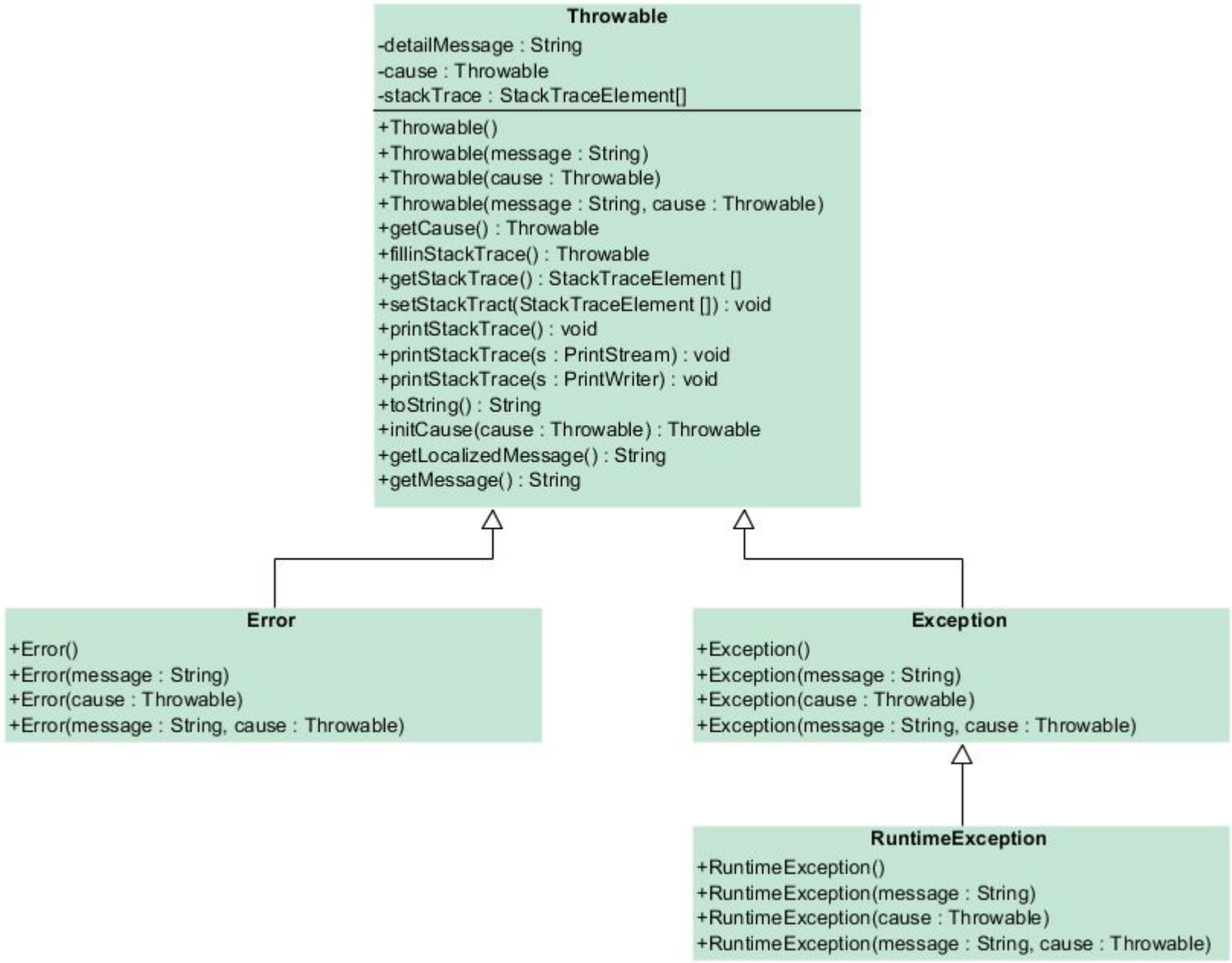

DIASOFT
всё по-настоящему

DIASOFT
всё по-настоящему

VI. Исключения

2. Классы исключений


Throwable
-detailMessage : String
-cause : Throwable
-stackTrace : StackTraceElement[]
+Throwable()
+Throwable(message : String)
+Throwable(cause : Throwable)
+Throwable(message : String, cause : Throwable)
+getCause() : Throwable
+fillinStackTrace() : Throwable
+getStackTrace() : StackTraceElement []
+setStackTract(StackTraceElement []) : void
+printStackTrace() : void
+printStackTrace(s : PrintStream) : void
+printStackTrace(s : PrintWriter) : void
+toString() : String
+initCause(cause : Throwable) : Throwable
+getLocalizedMessage() : String
+getMessage() : String
Error
+Error()
+Error(message : String)
+Error(cause : Throwable)
+Error(message : String, cause : Throwable)
Exception
+Exception()
+Exception(message : String)
+Exception(cause : Throwable)
+Exception(message : String, cause : Throwable)
RuntimeException
+RuntimeException()
+RuntimeException(message : String)
+RuntimeException(cause : Throwable)
+RuntimeException(message : String, cause : Throwable)

Класс Throwable

DIASOFT
всё по-настоящему

```
public class Throwable implements Serializable {

    private String detailMessage;

    public Throwable() {
        fillInStackTrace();
    }

    public Throwable(String message) {
        fillInStackTrace();
        detailMessage = message;
    }

    public String getMessage() {
        return detailMessage;
    }

    public String getLocalizedMessage() {
        return getMessage();
    }

    public String toString() {
        String s = getClass().getName();
        String message = getLocalizedMessage();
        return (message != null) ? (s + ": " + message) : s;
    }
}
```

Получение информации об исключении

```
public class ExceptionMethodsDemo {
    public static void main(String[] args) {
        try {
            throw new Exception("My Exception");
        } catch (Exception e) {

            System.out.println("Caught Exception");
            System.out.println("getMessage():" + e.getMessage());
            System.out.println("getLocalizedMessage():"
                    + e.getLocalizedMessage());
            System.out.println("toString():" + e);
        }
    }
}
```

```
Caught Exception
getMessage():My Exception
getLocalizedMessage():My Exception
toString():java.lang.Exception: My Exception
```

Стек вызовов

	Трассировка стека предоставляет информацию об истории выполнения текущего потока и даёт список методов которые вызывались до выбрасывания исключения. Нулевой элемент представляет вершину стека, то есть последний вызванный метод последовательности – метод в котором был создан и выброшен объект Throwable. Последний элемент массива представляет низ стека, то есть первый вызванный метод последовательности.

Класс Throwable

DIASOFT
всё по-настоящему

```
public class Throwable implements Serializable {

    private StackTraceElement[] stackTrace;

    public void printStackTrace() {
        printStackTrace(System.err);
    }

    public void printStackTrace(PrintStream s) {
        synchronized (s) {
            s.println(this);
            StackTraceElement[] trace = getOurStackTrace();
            for (int i=0; i < trace.length; i++)
                s.println("\tat " + trace[i]);

            Throwable ourCause = getCause();
            if (ourCause != null)
                ourCause.printStackTraceAsCause(s, trace);
        }
    }

    public StackTraceElement[] getStackTrace() {
        return (StackTraceElement[]) getOurStackTrace().clone();
    }

    private synchronized StackTraceElement[] getOurStackTrace() {

        if (stackTrace == null) {
            int depth = getStackTraceDepth();
            stackTrace = new StackTraceElement[depth];
            for (int i=0; i < depth; i++)
                stackTrace[i] = getStackTraceElement(i);
        }
        return stackTrace;
    }

    native int getStackTraceDepth();

}
```

Стек вызовов

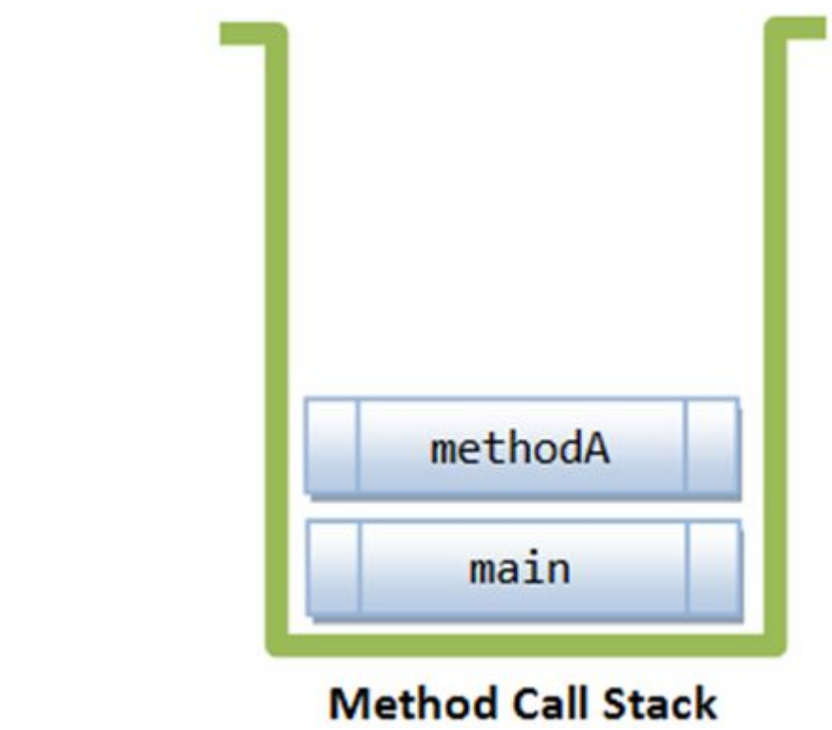
```
public class StackTraceDemo {

    static void methodA() throws Exception {
        throw new Exception();
    }

    static void methodB() throws Exception {
        methodA();
    }

    static void methodC() throws Exception {
        methodB();
    }

    ...
}
```

Стек вызовов

DIASOFT
всё по-настоящему

```
public class StackTraceDemo {

    ...

    public static void main(String[] args) {
        try {
            methodA();
        } catch (Exception e) {
            for (StackTraceElement ste : e.getStackTrace())
                System.out.println(ste.getMethodName());
        }

        System.out.println("--------------------------------");
        try {
            methodB();
        } catch (Exception e) {
            for (StackTraceElement ste : e.getStackTrace())
                System.out.println(ste.getMethodName());
        }

        System.out.println("--------------------------------");
        try {
            methodC();
        } catch (Exception e) {
            for (StackTraceElement ste : e.getStackTrace())
                System.out.println(ste.getMethodName());
        }
    }
}
```

Стек вызовов



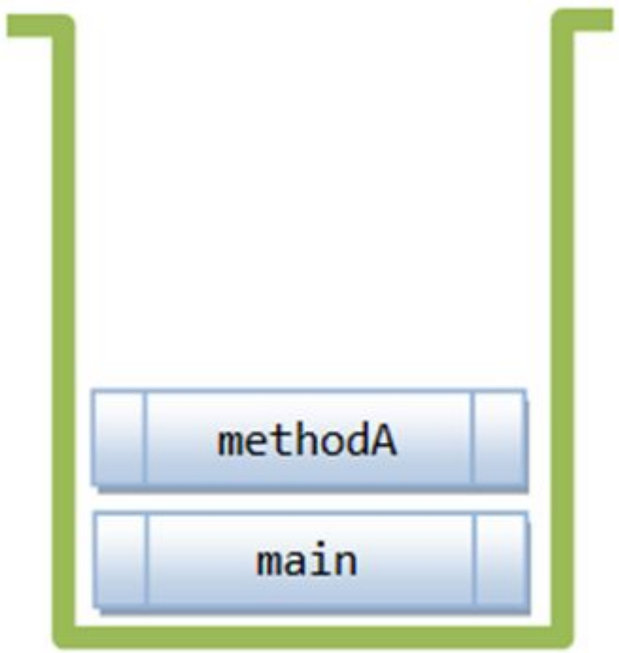

```
methodA
main
--------------------------------
methodA
methodB
main
--------------------------------
methodA
methodB
methodC
main
```

Стек вызовов

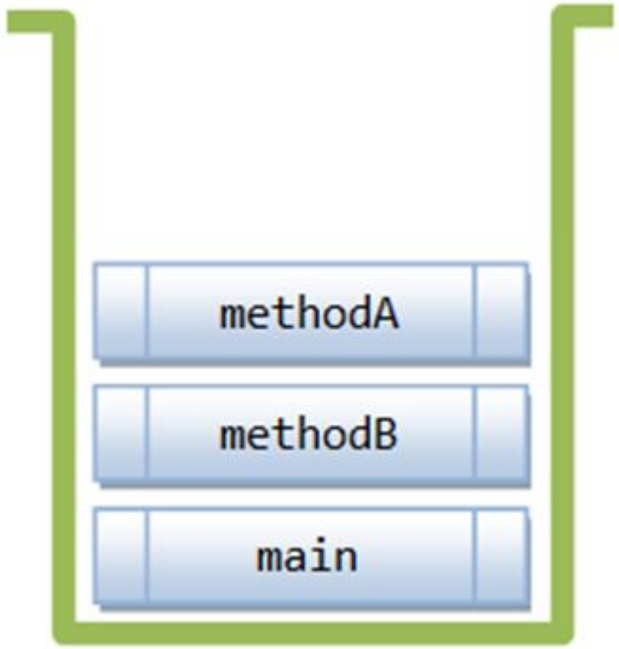
DIASOFT
всё по-настоящему

```
public class AnotherStackTraceDemo {

    ...

     public static void main(String[] args) {
        try {
            methodA();
        } catch (Exception e) {
            e.printStackTrace(System.out);
        }

        System.out.println("--------------------------------");
        try {
            methodB();
        } catch (Exception e) {
            e.printStackTrace(System.out);
        }

        System.out.println("--------------------------------");
        try {
            methodC();
        } catch (Exception e) {
            e.printStackTrace(System.out);
        }
    }
}
```

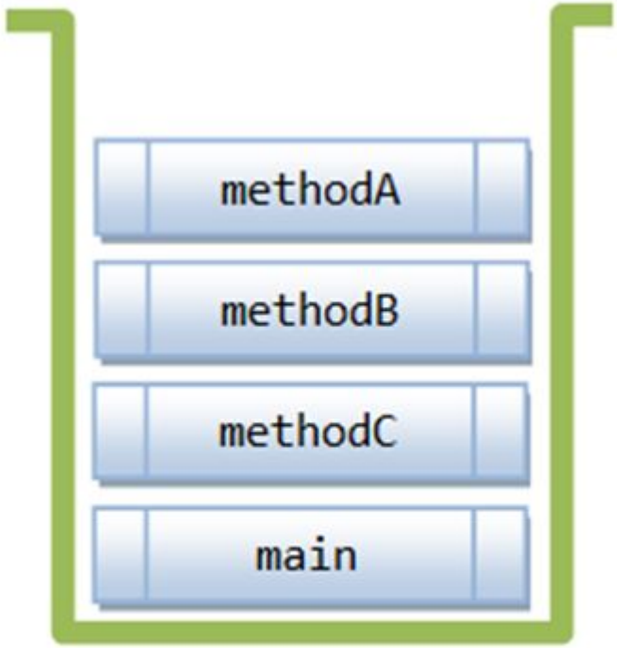
Стек вызовов



Method Call Stack



Method Call Stack



Method Call Stack

```
java.lang.Exception
at classes.AnotherStackTraceDemo.methodA(AnotherStackTraceDemo.java:6)
at classes.AnotherStackTraceDemo.main(AnotherStackTraceDemo.java:19)
--------------------------------
java.lang.Exception
at classes.AnotherStackTraceDemo.methodA(AnotherStackTraceDemo.java:6)
at classes.AnotherStackTraceDemo.methodB(AnotherStackTraceDemo.java:10)
at classes.AnotherStackTraceDemo.main(AnotherStackTraceDemo.java:26)
--------------------------------
java.lang.Exception
at classes.AnotherStackTraceDemo.methodA(AnotherStackTraceDemo.java:6)
at classes.AnotherStackTraceDemo.methodB(AnotherStackTraceDemo.java:10)
at classes.AnotherStackTraceDemo.methodC(AnotherStackTraceDemo.java:14)
at classes.AnotherStackTraceDemo.main(AnotherStackTraceDemo.java:33)
```

Элемент стека вызовов

Класс Throwable

DIASOFT
всё по-настоящему

```
public class Throwable implements Serializable {

    public void setStackTrace(StackTraceElement[] stackTrace) {
        StackTraceElement[] defensiveCopy =
            (StackTraceElement[]) stackTrace.clone();
        for (int i = 0; i < defensiveCopy.length; i++)
            if (defensiveCopy[i] == null)
                throw new NullPointerException("stackTrace[" + i + "]");

        this.stackTrace = defensiveCopy;
    }
}
```

Класс StackTraceElement

```
public class StackTraceElement implements java.io.Serializable {

    private String declaringClass;
    private String methodName;
    private String fileName;
    private int lineNumber;

    public boolean isNativeMethod() { return lineNumber == -2; }

    public StackTraceElement(String declaringClass, String methodName, String fileName, int lineNumber) {
        if (declaringClass == null)
            throw new NullPointerException("Declaring class is null");
        if (methodName == null)
            throw new NullPointerException("Method name is null");

        this.declaringClass = declaringClass;
        this.methodName     = methodName;
        this.fileName       = fileName;
        this.lineNumber     = lineNumber;
    }

     public String toString() {
        return getClassName() + "." + methodName +
            (isNativeMethod() ? "(Native Method)" :
             (fileName != null && lineNumber >= 0 ?
              "(" + fileName + ":" + lineNumber + ")" :
              (fileName != null ?  "("+fileName+")" : "(Unknown Source)")));
    }
    ...
}
```

C	Класс StackTraceElement используется для представления стекового кадра в трассировке стека. Стековый кадр на вершине стека представляет точку в которой трассировка стека была создана. Остальные стековые кадры представляют вызовы методов. В классе есть поля для имени класса вызванного метода, названия метода, имени файла и строки в файле.

Элемент стека вызовов

```
public class StackTraceElementDemo {

    public static void main(String[] args) {

        try {
            methodA();
        } catch (Throwable e) {

            e.printStackTrace();
        }
    }

    public static void methodA() throws RuntimeException {

        RuntimeException t = new RuntimeException("This is a new Exception...");
        StackTraceElement[] trace = new StackTraceElement[] { new StackTraceElement(
                "ClassName", "methodName", "fileName", 5) };

        t.setStackTrace(trace);
        throw t;
    }
}
```

```
java.lang.RuntimeException: This is a new Exception...
at ClassName.methodName(fileName:5)
```

Перебрасывание исключений из блока catch

Во время обработки исключения блоком catch обрабатываемое исключение может быть переброшено. В этом случае трассировка стека будет сохранена если специально её не менять. Такой подход может иногда использоваться для локального логирования.

Перебрасывание исключений

DIASOFT
всё по-настоящему

```
public class RethrowingDemo {

    public static void methodA() throws Exception {
        System.out.println("Originating the exception in methodA()");
        throw new Exception("Thrown from methodA()");
    }

    public static void methodB() throws Exception {
        try {
            methodA();
        } catch (Exception e) {
            System.out.println("Inside methodB");
            e.printStackTrace(System.out);
            throw e;
        }
    }

    public static void methodC() throws Exception {
        try {
            methodA();
        } catch (Exception e) {
            System.out.println("Inside methodC");
            e.printStackTrace(System.out);
            throw (Exception) e.fillInStackTrace();
        }
    }
}
```

Перебрасывание исключений

DIASOFT
всё по-настоящему

```
public class RethrowingDemo {

    public static void main(String[] args) {

        try {
            methodB();
        } catch (Exception e) {
            System.out.println("Caught in main");
            e.printStackTrace(System.out);
        }


        System.out.println();


        try {
            methodC();
        } catch (Exception e) {
            System.out.println("Caught in main");
            e.printStackTrace(System.out);
        }
    }

}
```

Перебрасывание исключений

```
Originating the exception in methodA()
Inside methodB
java.lang.Exception: Thrown from methodA()
at classes.RethrowingDemo.methodA(RethrowingDemo.java:7)
at classes.RethrowingDemo.methodB(RethrowingDemo.java:12)
at classes.RethrowingDemo.main(RethrowingDemo.java:32)
Caught in main
java.lang.Exception: Thrown from methodA()
at classes.RethrowingDemo.methodA(RethrowingDemo.java:7)
at classes.RethrowingDemo.methodB(RethrowingDemo.java:12)
at classes.RethrowingDemo.main(RethrowingDemo.java:32)

Originating the exception in methodA()
Inside methodC
java.lang.Exception: Thrown from methodA()
at classes.RethrowingDemo.methodA(RethrowingDemo.java:7)
at classes.RethrowingDemo.methodC(RethrowingDemo.java:22)
at classes.RethrowingDemo.main(RethrowingDemo.java:38)
Caught in main
java.lang.Exception: Thrown from methodA()
at classes.RethrowingDemo.methodC(RethrowingDemo.java:26)
at classes.RethrowingDemo.main(RethrowingDemo.java:38)
```

Сцепленные исключения

Собирание исключений в цепочку или оборачивание исключений позволяет ассоциировать с одним исключением другое которое описывает причину его появления. Такой подход позволяет перебрасывать перехваченное исключение обернув его другим исключением. Таким образом можно добиться того чтобы метод выбрасывал исключения определённые на том же самом уровне абстракции, но без потери информации с нижних уровней. Кроме того оборачивание исключений позволяет обернуть контролируемое исключение неконтролируемым если программа не может восстановиться после выбрасывания контролируемого исключения.

Не сцепленные исключения

DIASOFT
всё по-настоящему

```
public class NoChainedExceptionDemo {

    public static void methodA() throws Exception {
        System.out.println("Originating the exception in methodA()");
        throw new Exception("Thrown from methodA()");
    }

    public static void methodB() throws Exception {
        try {
            methodA();
        } catch (Exception e) {
            System.out.println("Throwing new exception in methodB()");
            throw new Exception("Thrown from methodB()");
        }
    }

    public static void methodC() {
        try {
            methodB();
        } catch (Exception e) {
            System.out.println("Throwing new exception in methodC()");
            throw new RuntimeException("Thrown from methodC()");
        }
    }

    public static void main(String[] args) {
            methodC();
    }
}
```

Не сцепленные исключения

```
Originating the exception in methodA()
Throwing new exception in methodB()
Throwing new exception in methodC()
Exception in thread "main" java.lang.RuntimeException: Thrown from methodC()
at classes.NoChainedExceptionDemo.methodC(NoChainedExceptionDemo.java:27)
at classes.NoChainedExceptionDemo.main(NoChainedExceptionDemo.java:32)
```

Класс Throwable

```
public class Throwable implements Serializable {

    private Throwable cause = this;

    public Throwable getCause() {
        return (cause==this ? null : cause);
    }

    public Throwable(String message, Throwable cause) {
        fillInStackTrace();
        detailMessage = message;
        this.cause = cause;
    }

    public synchronized Throwable initCause(Throwable cause) {
        if (this.cause != this)
            throw new IllegalStateException("Can't overwrite cause");
        if (cause == this)
            throw new IllegalArgumentException("Self-causation not permitted");
        this.cause = cause;
        return this;
    }
}
```

C	Класс Throwable включает поле cause типа Throwable для хранения ссылки на другое исключение которое является его причиной. Причину исключения можно задать либо в конструкторе или с помощью метода initCause(Throwable). Классы потомки Throwable могут определить конструктор принимающий Throwable и делегирующий (возможно не напрямую) задание причины одному из конструкторов Throwable. При выводе трассировки стека исключения выводится и трассировка стека для исключения которое являлось его причиной.

Класс Throwable

```
public class Throwable implements Serializable {

    public void printStackTrace(PrintStream s) {
        synchronized (s) {
            s.println(this);
            StackTraceElement[] trace = getOurStackTrace();
            for (int i=0; i < trace.length; i++)
                s.println("\tat " + trace[i]);

            Throwable ourCause = getCause();
            if (ourCause != null)
                ourCause.printStackTraceAsCause(s, trace);
        }
    }

    private void printStackTraceAsCause(PrintStream s, StackTraceElement[] causedTrace) {

        StackTraceElement[] trace = getOurStackTrace();
        int m = trace.length-1, n = causedTrace.length-1;
        while (m >= 0 && n >=0 && trace[m].equals(causedTrace[n])) {
            m--; n--;
        }
        int framesInCommon = trace.length - 1 - m;

        s.println("Caused by: " + this);
        for (int i=0; i <= m; i++)
            s.println("\tat " + trace[i]);
        if (framesInCommon != 0)
            s.println("\t... " + framesInCommon + " more");

        Throwable ourCause = getCause();
        if (ourCause != null)
            ourCause.printStackTraceAsCause(s, trace);
    }

    public Throwable getCause() {
        return (cause==this ? null : cause);
    }
}
```

Сцепленные исключения

DIASOFT
всё по-настоящему

```
public class ChainedExceptionDemo {

    public static void methodA() throws Exception {
        System.out.println("Originating the exception in methodA()");
        throw new Exception("Thrown from methodA()");
    }

    public static void methodB() throws Exception {
        try {
            methodA();
        } catch (Exception e) {
            System.out.println("Throwing new exception in methodB()");
            throw new Exception("Thrown from methodB()", e);
        }
    }

    public static void methodC() {
        try {
            methodB();
        } catch (Exception e) {
            System.out.println("Throwing new exception in methodC()");
            throw new RuntimeException("Thrown from methodC()", e);
        }
    }

    public static void main(String[] args) {
        methodC();
    }
}
```

Сцепленные исключения

```
Originating the exception in methodA()
Throwing new exception in methodB()
Throwing new exception in methodC()
Exception in thread "main" java.lang.RuntimeException: Thrown from methodC()
at classes.ChainedExceptionDemo.methodC(ChainedExceptionDemo.java:27)
at classes.ChainedExceptionDemo.main(ChainedExceptionDemo.java:32)
Caused by: java.lang.Exception: Thrown from methodB()
at classes.ChainedExceptionDemo.methodB(ChainedExceptionDemo.java:18)
at classes.ChainedExceptionDemo.methodC(ChainedExceptionDemo.java:24)
... 1 more
Caused by: java.lang.Exception: Thrown from methodA()
at classes.ChainedExceptionDemo.methodA(ChainedExceptionDemo.java:10)
at classes.ChainedExceptionDemo.methodB(ChainedExceptionDemo.java:15)
... 2 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