



Исключения

Лекция 4

Повторение



- Что означают ключевые слова:
 - ❖ super,
 - ❖ this?
- Что такое статические члены класса?
- Что такое абстрактные классы?
- Что такое интерфейсы?
- Какие виды внутренних классов вы знаете?
- Что такое классы-оболочки? Для чего они необходимы?

Цели

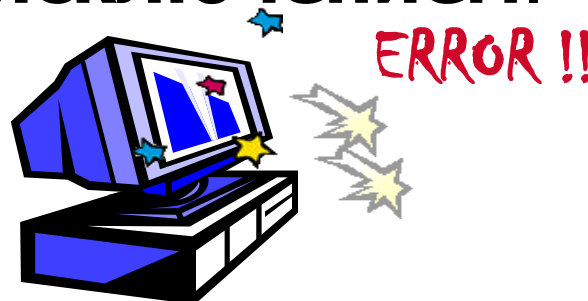
- Дать определение исключения.
- Описать обработку исключений.
- Описать блоки `try`, `catch` и `finally`.
- Объяснить использование нескольких блоков `catch`.
- Использовать вложенные блоки `try` / `catch`.
- Объяснить использование ключевых слов `throw` и `throws`.
- Создавать исключения, определённые пользователем.

Что такое исключение?

- Если во время выполнения программы возникает ошибка, то это называют **исключительной ситуацией** или просто **исключением**.

Пример:

$$4 / 0 =$$

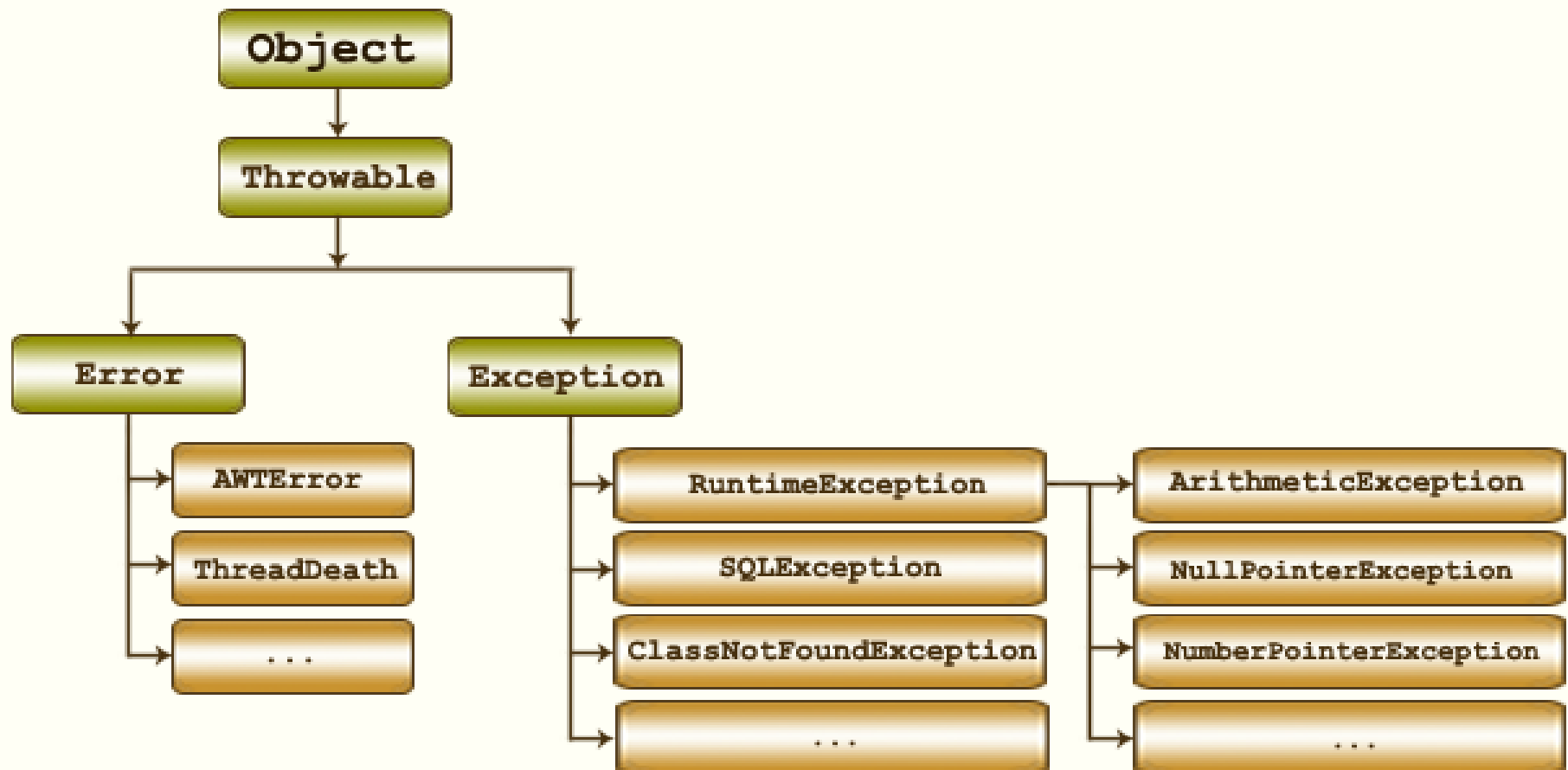


- При возникновении исключения программа немедленно завершается, и управление возвращается операционной системе.
- Обработка исключений выполняется с целью идентификации ошибок и их перехвата.

Обработка исключений

- При возникновении исключительной ситуации исполняющая система **создает объект** определенного класса, соответствующего возникшей ситуации, содержащий сведения о том, что, где и когда произошло.
- Этот объект передается на обработку программе, в которой возникло исключение.
- Если программа не обрабатывает исключение, то объект возвращается обработчику по умолчанию исполняющей системы.
- Обработчик поступает очень просто: выводит на консоль сообщение о произошедшем исключении и прекращает выполнение программы.

Иерархия классов исключений



Иерархия классов исключений

- В языке **Java** каждая исключительная ситуация реализуется как экземпляр класса **Throwable** или его наследников.
- У класса **Throwable** два непосредственных наследника — классы **Error** и **Exception**.
- Они не добавляют новых методов, а служат для разделения классов-исключений на два больших семейства — семейство классов-ошибок (**error**) и семейство классов-исключений (**exception**).
- Классы-ошибки, расширяющие класс **Error**, свидетельствуют о возникновении сложных ситуаций в виртуальной машине Java. Их обработка требует глубокого понимания всех тонкостей работы JVM. Ее не рекомендуется выполнять в обычной программе. Не советуют даже выбрасывать ошибки оператором **throw**. Не следует делать свои классы-исключения расширениями класса **Error** или какого-то его подкласса.
- Имена классов-ошибок, по соглашению, заканчиваются словом **Error**.

Иерархия классов исключений

- Классы-исключения, расширяющие класс **Exception**, отмечают возникновение обычной нештатной ситуации, которую можно и даже нужно обработать.
- Классов-исключений очень много, более двухсот.
- В большинстве случаев подбирается готовый класс-исключение для обработки исключительных ситуаций в своей программе.
- При желании можно создать и свой класс-исключение, расширив класс **Exception** или любой его подкласс.

Иерархия классов исключений

Исключение	Описание
Exception	Корневой класс в иерархии классов исключений
RuntimeException	Базовый класс для многих исключений <code>java.lang</code>
ArihmeticException	Арифметическая ошибка, например, деление на ноль
IllegalArgumentException	Метод принял недопустимый аргумент
ArrayIndexOutOfBoundsException	Индекс массива меньше 0 или больше, чем реальный размер массива
NullPointerException	Попытка доступа к члену объекта <code>null</code>
SecurityException	Условия безопасности не позволяют выполнить операцию
ClassNotFoundException	Невозможно загрузить требуемый класс
NumberFormatException	Некорректное преобразование строки в число с плавающей точкой
FileNotFoundException	Невозможно найти требуемый файл
EOFException	Достижение конца файла
NoSuchMethodException	Требуемый метод не существует

Обработка исключений

- В Java обработка исключений управляется посредством пяти ключевых слов: **try**, **catch**, **throw**, **throws** и **finally**.
- Инструкции программы, в которых должны отслеживаться исключения, помещаются в блок **try**.
- Используя ключевое слово **catch**, программист может перехватить исключение и обработать его некоторым разумным образом.
- Для генерации исключения «вручную» мы используем ключевое слово **throw**.
- Вариант **throws** используется в методе для указания на то, что этот метод будет генерировать исключения.
- В блоке **finally** можно определить код, который безусловно необходимо выполнить перед возвращением из метода.

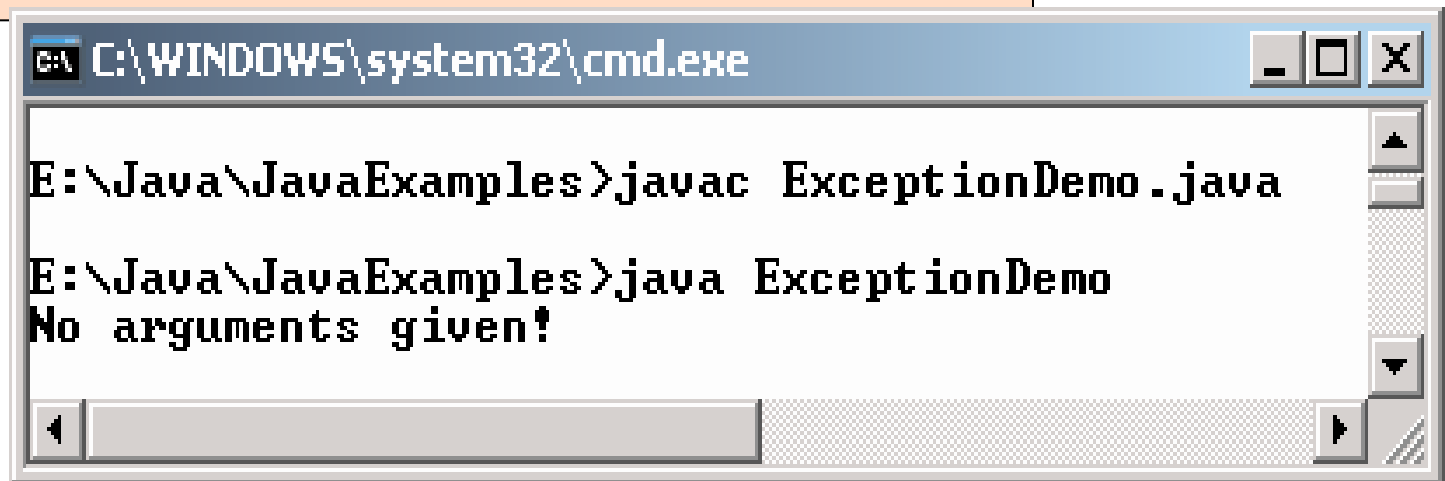
Модель обработки исключений

- Для того, чтобы попытаться (**try**) перехватить (**catch**) объект-исключение, надо весь код программы, в котором может возникнуть исключительная ситуация, охватить оператором **try{} catch() {}**.
- Каждый блок **catch()** перехватывает исключение только одного типа, того, который указан в его аргументе. Но можно написать несколько блоков **catch(){}** для перехвата нескольких типов исключений.

```
try {  
    // Здесь возможно возбуждение  
    // исключительной ситуации  
}  
catch (ТипИсключительнойСитуации)  
{  
    // Здесь производится обработка  
    // перехваченной исключительной ситуации  
}
```

Блоки try и catch

```
class ExceptionDemo
{
    public static void main(String args[]) {
        try {
            String text = args[0];
            System.out.println(text);
        }
        catch(Exception e) {
            System.out.println("No arguments given! ");
        }
    }
}
```



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The prompt is at "E:\Java\JavaExamples>". The user has entered "javac ExceptionDemo.java" and the output is empty. Then, the user has entered "java ExceptionDemo" and the output is "No arguments given!".

```
C:\WINDOWS\system32\cmd.exe

E:\Java\JavaExamples>javac ExceptionDemo.java

E:\Java\JavaExamples>java ExceptionDemo
No arguments given!
```

throws

- Обычно все методы, в которых может возникнуть исключительная ситуация, описываются особым образом.

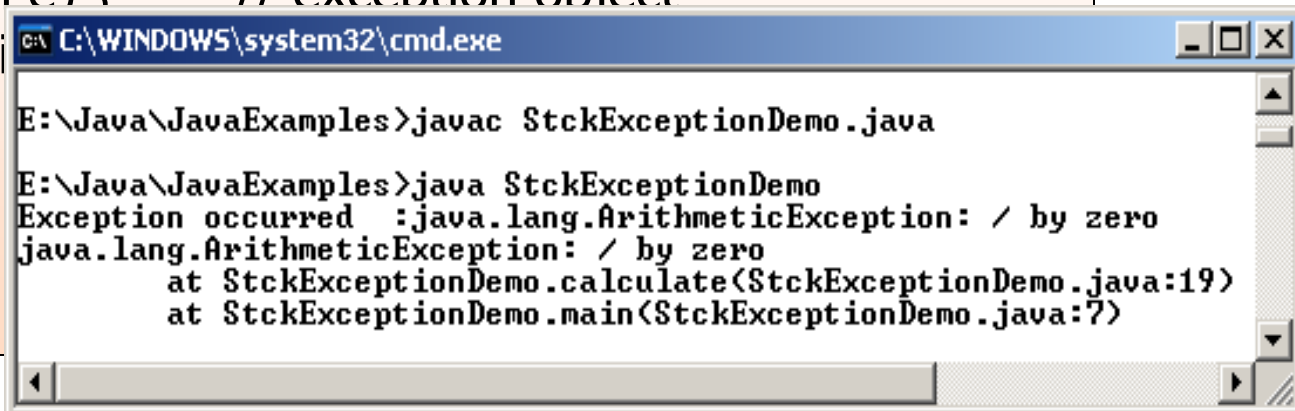
```
void SomeMethod () throws FileNotFoundException {  
    //тело метода  
}
```

- В этом описании оператор throws обозначает, что **метод потенциально может создать/вызвать исключительную ситуацию FileNotFoundException**, поскольку не найден какой-либо файл.
- Теперь любой вызов этого метода в программе должен быть обрамлен описанием блока **try-catch**, иначе компилятор выдаст ошибку и не обработает исходный текст вашей программы.

- Программы на языке Java могут самостоятельно возбуждать исключительные ситуации, используя для этого оператор **throw new ArithmeticException();**
- В точке метода, где встречается **throw**, выполнение метода прерывается и управление передается в тот метод, который вызвал ошибку.
- Если исключительная ситуация может быть обработана методом, то вызывается его обработчик.
- Если же это невозможно, то поток управления передается дальше, и так происходит до того момента, когда исключительная ситуация не будет перехвачена или пока ее не перехватит виртуальная машина Java.
- В последнем случае выполнение программы прерывается и выводится сообщение об ошибке.

Обработка исключений

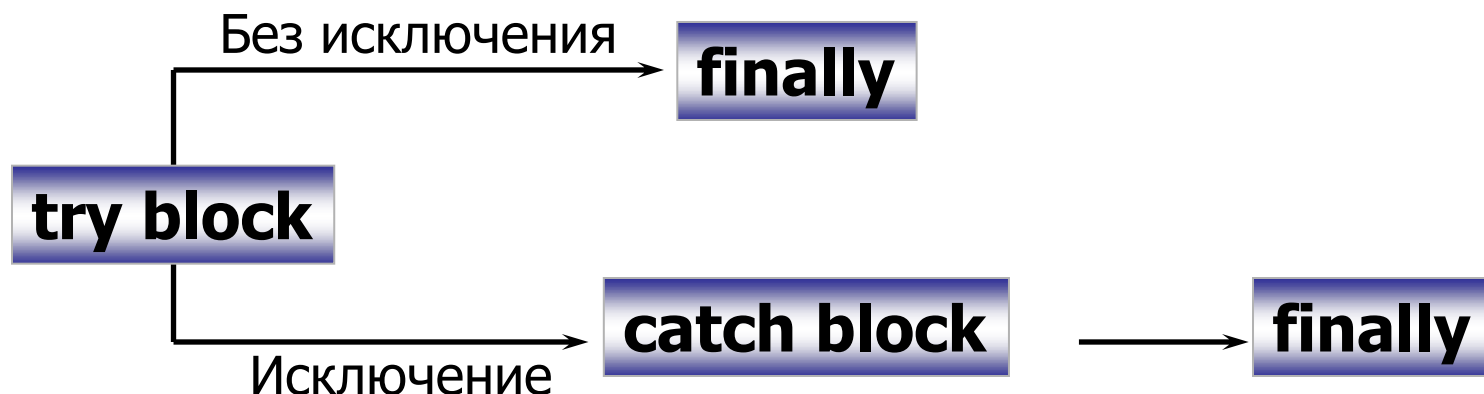
```
public class StckExceptionDemo {  
  
    static int calculate( int no, int no1) {  
        int num = no / no1;  
        return num;  
    }  
    public static void main(String args[]) {  
        try {  
            int num = calculate(9,0); // user defined method  
            System.out.println(num);  
        }  
        catch(Exception e) { // exception object  
            System.err.println(e);  
        }  
    }  
}
```



```
C:\WINDOWS\system32\cmd.exe  
E:\Java\JavaExamples>javac StckExceptionDemo.java  
E:\Java\JavaExamples>java StckExceptionDemo  
Exception occurred : java.lang.ArithmeticException: / by zero  
java.lang.ArithmeticException: / by zero  
    at StckExceptionDemo.calculate(StckExceptionDemo.java:19)  
    at StckExceptionDemo.main(StckExceptionDemo.java:7)
```

Блок **finally**

- Гарантирует, что при возникновении исключения вся работа по очистке памяти будет выполнена.
- Используется в сочетании с блоком **try**.
- Гарантируется выполнение независимо от того, возникло исключение или нет.
- При генерации исключения блок **finally** отработает даже в том случае, если отсутствует соответствующий блок **catch**.



Несколько блоков **catch**

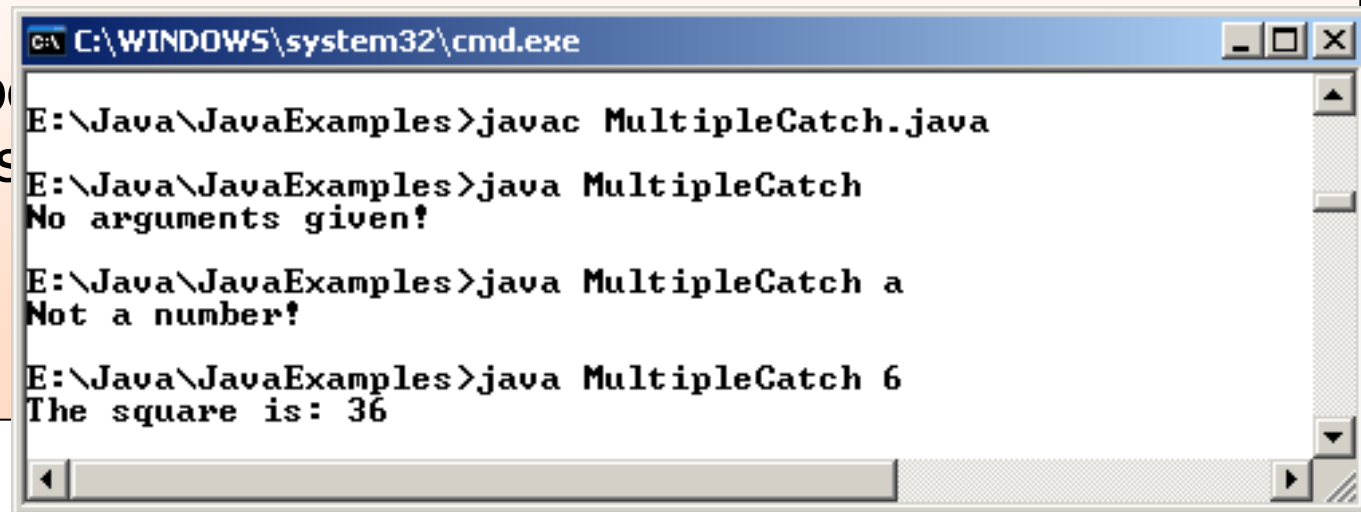
- Один фрагмент кода может генерировать несколько ошибок.
- Поэтому возможно наличие нескольких блоков **catch**.
- Порядок выполнения инструкций **catch** соответствует порядку их расположения.

```
.....  
try{  
}  
catch(ArrayIndexOutOfBoundsException e) {  
}  
catch(Exception e) {  
}  
.....
```

- `ArrayIndexOutOfBoundsException`, являющийся подклассом класса `Exception`, обрабатывается первым.

Пример

```
class MultipleCatch {  
    public static void main(String args[]) {  
        try {  
            String num = args[0];  
            int numValue = Integer.parseInt(num);  
            System.out.println("The square is: " + numValue * numValue);  
        }  
        catch(ArrayIndexOutOfBoundsException ne) {  
            System.out.println("No arguments given!");  
        }  
        catch(NumberFormatException ne) {  
            System.out.println("Not a number!");  
        }  
    }  
}
```



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window displays the following commands and their outputs:

```
E:\Java\JavaExamples>javac MultipleCatch.java  
E:\Java\JavaExamples>java MultipleCatch  
No arguments given!  
E:\Java\JavaExamples>java MultipleCatch a  
Not a number!  
E:\Java\JavaExamples>java MultipleCatch 6  
The square is: 36
```

Использование throw и throws

- Исключения генерируются с использованием ключевого слова `throw`.

```
try {  
    if(flag<0) {  
        throw new NullPointerException();  
    }  
}
```

- Один метод может генерировать (**throw**) более одного исключения.

Использование throws

```
public class Example {  
    void check() throws NullPointerException, NegativeArraySizeException {  
        if (flag < 0)  
            throw new NullPointerException();  
        if(arrsize < 0)  
            throw new NegativeArraySizeException();  
    }  
  
    void exceptionExample() {  
        try {  
            check();  
        }  
        catch(Exception e){  
            //действия ....  
        }  
    }  
}
```

- Метод **check()** включает ключевое слово **throws**
- Такие методы должны вызываться внутри блоков **try / catch**

Исключения, определённые пользователем

- Встроенные исключения не всегда достаточны для перехвата всех ошибок.
- Поэтому возникает необходимость в классе исключения, определяемого пользователем.
- Он должен быть подклассом класса **Exception**.
- Новый тип исключения может быть перехвачен отдельно от прочих подклассов из группы **Throwable**.
- Определённые пользователем классы исключений, которые создаются, наследуют все методы класса **Throwable**.