Язык программирования Java

Мультимедийный курс

автор: Васильев А.Н.

www.vasilev.kiev.ua

Киев 2017

Обработка исключительных ситуаций

- Перехват и обработка ошибок
- Принципы обработки исключений
- Вложенные try-catch блоки
- Использование объекта исключения
- Генерирование исключений
- Контролируемые и неконтролируемые исключения
- Создание пользовательских классов исключений

Перехват и обработка ошибок

Основу системы обработки исключительных ситуаций составляет иерархия классов, описывающих исключительные ситуации, и оператор `try-catch`

Пример обработки ошибок - 1

```
// Статический импорт:
import static javax.swing.JOptionPane.*;
import static java.lang.Integer.*;
// Главный класс:
class TryCatchExampleDemo{
   // Главный метод:
   public static void main(String[] args){
      // Текстовая переменная для записи
      // считываемого значения:
      String input;
      // Переменная для записи целого числа:
      int num;
      // Отображение диалогового кона с полем ввода:
      input=showInputDialog(null,
         "Введите число", // Текст над полем ввода
         "Число",         // Название окна
         QUESTION_MESSAGE // Тип окна
      );
      // Продолжение на следующем слайде!!!
```

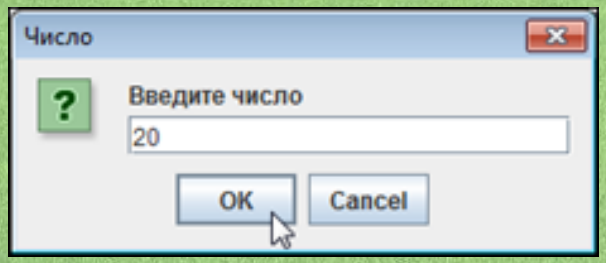
Перехват и обработка ошибок

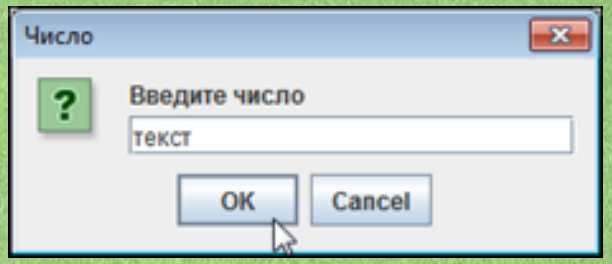
Пример обработки ошибок - 2

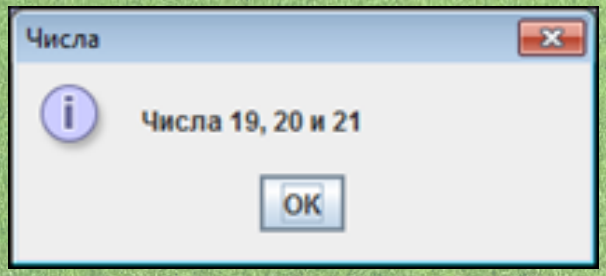
```
        // Блок контролируемого кода:
        try{
            // Попытка преобразовать текст в число:
            num=parseInt(input);
            // Отображение диалогового кона с числами:
            showMessageDialog(null,
                // Отображаемый в окне текст:
                "Числа "+(num-1)+", "+num+" и "+(num+1),
                "Числа",              // Название окна
                INFORMATION_MESSAGE   // Тип окна
            );
        // Обработка ошибок (код выполняется,
        // если в контрольном блоке возникла ошибка):
        }catch(Exception e){
            // Отображение диалогового окна:
            showMessageDialog(null,
                "Что-то пошло не так...", // Сообщение в окне
                "Ошибка",                 // Название окна
                ERROR_MESSAGE             // Тип окна
            );
        }
    }
}
```

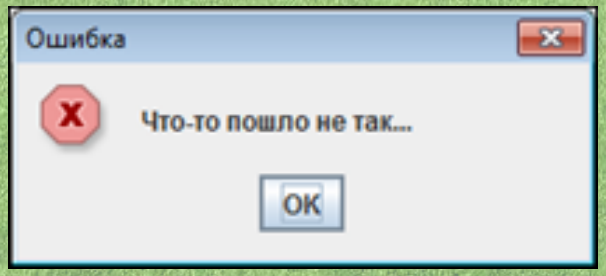
Перехват и обработка ошибок

Пример обработки ошибок - результат выполнения программы









Принципы обработки исключений

- В Java есть иерархия классов, соответствующих ошибкам определенного типа. В вершине иерархии находится класс `Throwable`
- У класса `Throwable` имеется два подкласса: `Error` и `Exception`. Класс `Error` соответствует "фатальным" ошибкам, которые обычно программными методами не обрабатываются
- У класса `Exception` много подклассов, и среди них есть класс `RuntimeException`
- Класс `RuntimeException` является суперклассом для многих классов ошибок

- Если ошибка возникла, то в соответствии с ее типом создается объект (исключение), который передается для обработки методу, который вызвал ошибку (говорят, что исключение вбрасывается в метод)
- Если в методе ошибка обрабатывается - хорошо. Если ошибка в методе не обрабатывается, то объект ошибки передается для обработки в метод, в котором вызывался данный метод, и так по цепочке, вплоть до главного метода программы
- Если и в главном методе ошибка не обрабатывается, то в дело вступает так называемый обработчик по умолчанию - программа досрочно прекращает выполнение

Принципы обработки исключений

- Фрагмент кода, который может вызвать ошибку, помещается в `try`-блок
- После `try`-блока следуют `catch`-блоки (их может быть несколько)
- Каждый `catch`-блок предназначен для обработки ошибок определенного типа. Под типом ошибки в данном случае имеется в виду класс, на основе которого, в случае возникновения ошибки, создается объект ошибки
- В каждом `catch`-блоке в круглых скобках кроме класса ошибки указывается формальное название для объекта ошибки. При необходимости, данный объект может быть использован в явном виде в программном коде `catch`-блока
- Если при выполнении программного кода в `try`-блоке ошибка не возникает, то все `catch`-блоки игнорируются
- Если в `try`-блоке возникла ошибка, то для данной ошибки создается объект, и далее с целью обработки исключения проверяются `catch`-блоки. Проверка выполняется на предмет совпадения класса объекта ошибки и класса, указанного в `catch`-блоке. Если совпадение найдено, объект ошибки передается в соответствующий `catch`-блок для обработки

Принципы обработки исключений

- Важное обстоятельство состоит в том, что "совпадение" понимается в широком смысле: классы могут совпадать не буквально достаточно, чтобы класс объекта ошибки был подклассом класса, указанного в `catch`-блоке. В этом смысле, если мы в `catch`-блоке указываем имя класса `Exception`, то такой `catch`-блок будет перехватывать фактически все ошибки (которые в принципе можно обработать), поскольку классы ошибок, с которыми мы имеем дело, прямо или опосредованно являются наследниками класса `Exception`

- В `try-catch` конструкции после `catch`-блоков может использоваться `finally`-блок (блок начинается ключевым словом `finally` и заключается в фигурные скобки). Программный код из `finally`-блока выполняется в любом случае: и если возникла ошибка, и если она не возникла. Обычно `finally`-блоки полезны при использовании вложенных `try`-блоков (когда внутри `try`-блока размещается еще одна `try-catch` конструкция)

Некоторые классы ошибок

Подклассы класса `RuntimeException`

Класс исключения	Описание
ArithmeticException	Ошибка, связанная с выполнением арифметических операций
ArrayStoreException	Ошибка, связанная с попыткой записать в массив объект типа, не соответствующего типу элементов массива
ClassCastException	Ошибка, связанная с попыткой некорректного приведения типов
IllegalArgumentException	Ошибка, связанная с некорректной передачей аргументов методу. У данного класса довольно много подклассов. Среди них, в частности, есть класс **NumberFormatException**, соответствующий ошибке, возникающей при попытке преобразовать некорректное текстовое представление для числа в число

Некоторые классы ошибок

Класс исключения	Описание
IndexOutOfBoundsException	Ошибка, связанная с некорректно указанным индексом. У данного класса есть два подкласса: класс **IndexOutOfBoundsException** соответствует ошибке, возникающей при неправильно указанном индексе массива, а класс **StringIndexOutOfBoundsException** соответствует ошибке, связанной с некорректно указанным индексом элемента в текстовой строке
NegativeArraySizeException	Ошибка, связанная с попыткой создать массив отрицательного размера
NoSuchElementException	Ошибка, связанная с обращением к несуществующему элементу
NullPointerException	Ошибка, связанная с использованием пустой ссылки (значение null) вместо ссылки на объект
UnsupportedOperationException	Ошибка связана с тем, что запрашиваемая операция не поддерживается

Программа: постановка задачи

В программе отображается диалоговое окно, в котором пользователю предлагается указать размер числового массива

Если пользователь вводит корректное значение, то создается соответствующий числовой массив и заполняется символами

Последовательность символов из массива отображается в диалоговом окне

После этого пользователя просят указать индекс элемента в массиве, и после считывания введенного пользователем индекса отображается соответствующий символ

Также в программе отслеживаются возможные ошибки, связанные с тем, что:

- пользователь отменил вод числа или ввел нечисловое значение;
- ввел отрицательное значение для размера массива;
- указал индекс элемента массива, который выходит за допустимый диапазон значений индекса

Программа

Обработка исключительных ситуаций - 1

```
// Статический импорт:
import static javax.swing.JOptionPane.*;
import static java.lang.Integer.*;
// Главный класс:
class MoreTryCatchDemo{
   public static void main(String[] args){
      // Переменная для считывания данных из поля ввода:
      String input;
      // Переменная массива:
      char[] symbs;
      // Переменные для записи размера массива
      // и индекса элемента:
      int size,index;
      // Отображение окна с полем ввода для считывания
      // размера массива:
      input=showInputDialog(null,
         "Укажите размер массива", // Текст над полем ввода
         "Символьный массив",      // Название окна
         QUESTION_MESSAGE          // Тип пиктограммы
      );
      // Контролируемый код:
      try{
         // Преобразование текста в число:
         size=parseInt(input);
         // Продолжение на следующем слайде!!!
```

Программа

Обработка исключительных ситуаций - 2

```
// Создание массива:
symbs=new char[size];
// Текстовая переменная для формирования
// текста сообщения:
String txt="| ";
// Заполнение массива символами и формирование
// текста для отображения в окне:
for(int k=0;k<size;k++){
   // Элементу массива присваивается значение:
   symbs[k]=(char)('A'+k);
   // К тексту дописывается значение элемента:
   txt+=symbs[k]+" | ";
} // Отображение сообщения с символами из массива:
showMessageDialog(null,
   txt,                    // Сообщение
   "Символы из массива", // Название окна
   INFORMATION_MESSAGE    // Тип сообщения
); // Отображение окна с полем ввода для считывания
// значения индекса элемента массива:
input=showInputDialog(null,
   // Текст над полем ввода:
   "Укажите индекс элемента",
   // Название окна:
   "Индекс элемента массива",
   // Тип пиктограммы:
   QUESTION_MESSAGE);      // Продолжение на следующем слайде...
```

Программа

Обработка исключительных ситуаций - 3

```
        // Преобразование текста в число:
        index=parseInt(input);
        // Текст для отображения в диалоговом окне:
        txt="Индекс - "+index+"\nСимвол - "+symbs[index];
        // Отображение значения элемента массива:
        showMessageDialog(null,
            txt,                  // Текст сообщения
            "Символ",             // Название окна
            INFORMATION_MESSAGE // Тип окна
        );
    }
    // Обработка исключительных ситуаций.
    // Пользователь отменил ввод или ввел
    // нечисловое значение:
    catch(NumberFormatException e){
        // Отображение окна с сообщением:
        showMessageDialog(null,
            // Сообщение:
            "К сожалению, возникала ошибка...",
            "Ошибка",          // Название окна
            WARNING_MESSAGE    // Тип окна
        );
    }
    // Продолжение на следующем слайде!!!
```

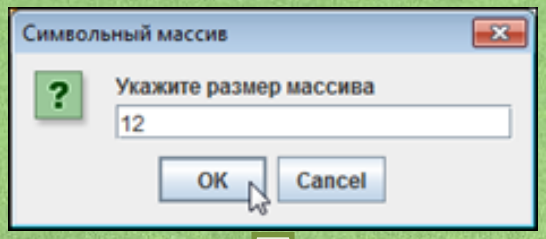
Программа

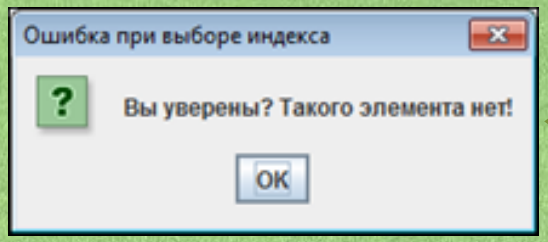
Обработка исключительных ситуаций - 4

```
        // Пользователь указал неверный размер для массива:
        catch(NegativeArraySizeException e){
            // Отображение диалогового окна:
            showMessageDialog(null,
                // Сообщение:
                "Некорректный размер массива!",
                // Название окна:
                "Ошибка при создании массива",
                // Пиктограмма:
                ERROR_MESSAGE
            );
        } // Пользователь указал неверное значение
        // для индекса элемента массива:
        catch(ArrayIndexOutOfBoundsException e){
            // Отображение диалогового окна:
            showMessageDialog(null,
                // Сообщение:
                "Вы уверены? Такого элемента нет!",
                // Название окна:
                "Ошибка при выборе индекса",
                // Пиктограмма:
                QUESTION_MESSAGE
            );
        }
    }
}
```

Программа

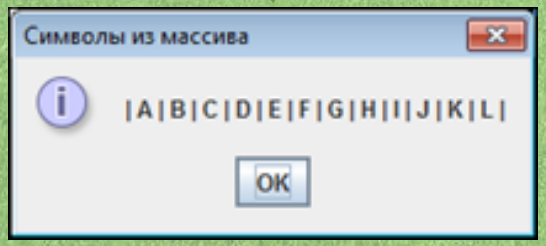
Обработка исключительных ситуаций - результат выполнения

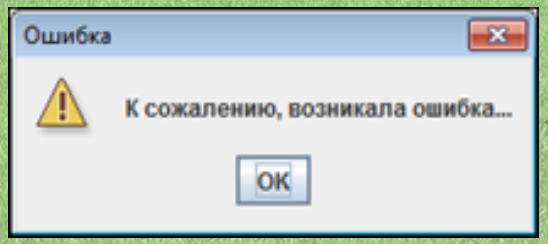




Введенный индекс выходит за допустимый диапазон

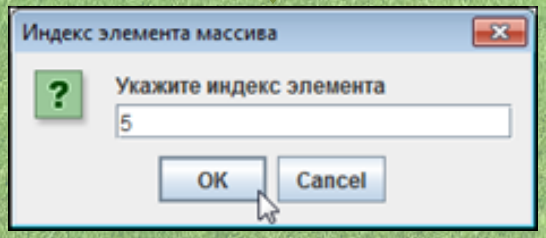
`ArrayIndexOutOfBoundsException`

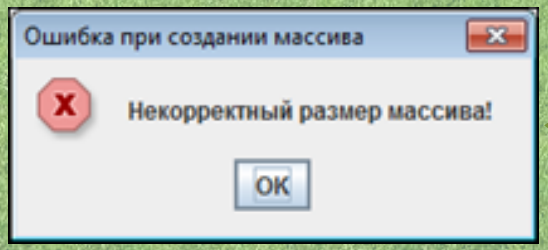




Введено нечисловое значение для индекса или отменен ввод

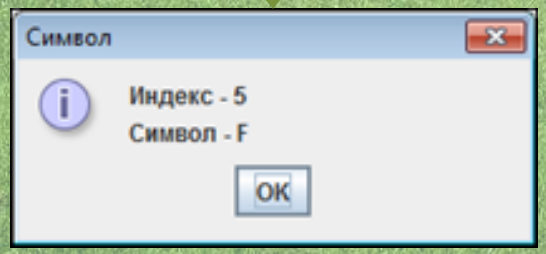
`NumberFormatException`





При определении размеров массива указано отрицательное число

`NegativeArraySizeException`



Вложенные блоки try-catch

Блоки `try-catch` могут быть вложенными. В таком случае если ошибка возникает во внутреннем `try`-блоке, то для обработки ошибки сначала просматриваются `catch`-блоки внутреннего `try`-блока (в котором возникла ошибка). Если ошибка внутренними `catch`-блоками не обрабатывается, то исключение передается для обработки во внешних `catch`-блоках

Вложенные блоки try-catch - 1

```
// Импорт классов:
import java.util.*;
// Главный класс:
class NestedTryCatchDemo{
   // Главный метод:
   public static void main(String[] args){
      // Объект класса Scanner (нужен для реализации
      // консольного ввода):
      Scanner input=new Scanner(System.in);
      // Переменная массива:
      char[] symbs;
      // Переменные для записи размера массива
      // и индекса элемента:
      int size,index;
      // Продолжение на следующем сайте!!!
```

Вложенные блоки try-catch

Вложенные блоки try-catch - 2

```
// Контролируемый код (внешний блок):
try{
   // Отображение сообщения:
   System.out.print("Укажите размер массива: ");
   // Считывание размера массива:
   size=input.nextInt();
   // Создание массива:
   symbs=new char[size];
   System.out.print("| ");
   // Заполнение массива символами:
   for(int k=0;k<size;k++){
      // Элементу массива присваивается значение:
      symbs[k]=(char)('A'+k);
      // Отображается значение элемента:
      System.out.print(symbs[k]+" | ");
   }
   // Контролируемый код (внутренний блок):
   try{
      System.out.print("\nУкажите индекс: ");
      // Считывание значения индекса:
      index=input.nextInt();
      // Значение элемента:
      System.out.println("Символ - "+symbs[index]);
   }
   // Продолжение на следующем слайде!!!
```

Вложенные блоки try-catch

Вложенные блоки try-catch - 3

```
            // Если пользователь ввел некорректный индекс:
            catch(ArrayIndexOutOfBoundsException e){
                System.out.println("Такого элемента нет");
            }
            // Блок выполняется всегда:
            finally{
                System.out.println("Массив создан успешно");
            }
            System.out.println("Для индекса указано числовое значение");
        }
        // Если введено не число:
        catch(InputMismatchException e){
            System.out.println("Ошибка: вы не ввели число");
        }
        // Если для массива указан отрицательный размер:
        catch(NegativeArraySizeException e){
            System.out.println("Неверный размер массива");
        }
        System.out.println("Выполнение программы завершено");
    }
}
```

Вложенные блоки try-catch

Детали

Класс `InputMismatchException` является подклассом класса `NoSuchElementException` (ошибка, связанная с тем, что запрашиваемый элемент не существует), который, в свою очередь, является подклассом класса `RuntimeException` (ошибки времени выполнения). Класс `NoSuchElementException` находится в пакете `java.util`, в то время как класс `RuntimeException` находится в пакете `java.lang`. Классы из пакета `java.lang` доступны автоматически даже без использования `import`-инструкции. Для использования классов из других пакетов, их приходится импортировать.

☑ На заметку

Код в `finally`-блоке выполняется в любом случае: и если при выполнении `try`-блока возникала ошибка, и если при выполнении `try`-блока ошибки не было. Блок `finally` в конструкции `try-catch` можно не использовать.

Вложенные блоки try-catch

Вложенные блоки try-catch - результат выполнения программы

```
Укажите размер массива: 12
| A | B | C | D | E | F | G | H | I | J | K | L |
Укажите индекс: 5
Символ - F
Массив создан успешно
Для индекса указано числовое значение
Выполнение программы завершено
```

Вложенные блоки try-catch

Вложенные блоки try-catch - результат выполнения программы

```
Укажите размер массива: 12
| A | B | C | D | E | F | G | H | I | J | K | L |
Укажите индекс: -5
Такого элемента нет
Массив создан успешно
Для индекса указано числовое значение
Выполнение программы завершено
```

ArrayIndexOutOfBoundsException

Вложенные блоки try-catch

Вложенные блоки try-catch - результат выполнения программы

```
Укажите размер массива: 12
| A | B | C | D | E | F | G | H | I | J | K | L |
Укажите индекс: пять
Массив создан успешно
Ошибка: вы не ввели число
Выполнение программы завершено
```

InputMismatchException

Вложенные блоки try-catch

Вложенные блоки try-catch - результат выполнения программы

```
Укажите размер массива: -5
Неверный размер массива
Выполнение программы завершено
```

NegativeArraySizeException

Вложенные блоки try-catch - результат выполнения программы

```
Укажите размер массива: пять
Ошибка: вы не ввели число
Выполнение программы завершено
```

InputMismatchException

☑ **На заметку**

Упоминавшийся ранее блок `finally` в данном случае не выполняется, поскольку он относится к внутренней `try-catch` конструкции.

Объект исключения

В описании `catch`-блоков неизменно присутствует, кроме имени класса обрабатываемого в блоке исключения, еще и формальное обозначение для объекта исключения. Эти объекты можно использовать при обработке ошибок. В частности, объект исключения можно использовать в выражениях в качестве "текстового" операнда: благодаря переопределенному методу `toString()` объект исключения в таких случаях автоматически приводится к текстовому формату

Программа: использование объекта исключения

В программе (в главном методе) последовательно размещено три `try-catch` конструкции. В `try`-блоке генерируется ошибка определенного типа, а в `catch`-блоке эта ошибка обрабатывается. Обработка ошибки состоит в том, что объект исключения (во всех трех `catch`-блоках обозначен как `e`) передается аргументом методу `println()`. Последствия приведения объекта исключения к текстовому формату можно оценить по результату выполнения программы

Объект исключения

Использование объекта исключения

```
class UsingExceptionObjectDemo{
   public static void main(String[] args){
      try{  // Код с ошибкой
         System.out.println("Отрицательный размер:");
         int[] a=new int[-1];
      } // Обработка ошибки:
      catch(NegativeArraySizeException e){
         System.out.println(e);
      } // Код с ошибкой:
      try{
         System.out.println("Неверный индекс:");
         int[] b={1};
         b[-1]=0;
      } // Обработка ошибки:
      catch(ArrayIndexOutOfBoundsException e){
         System.out.println(e);
      } // Код с ошибкой:
      try{
         System.out.println("Деление на нуль:");
         int c=10/0;
      } // Обработка ошибки:
      catch(ArithmeticException e){
         System.out.println(e);
      }
   }
}
```

Объект исключения

Использование объекта исключения - результат программы

```
Отрицательный размер:
java.lang.NegativeArraySizeException
Неверный индекс:
java.lang.ArrayIndexOutOfBoundsException: -1
Деление на нуль:
java.lang.ArithmeticException: / by zero
```

☑ На заметку

Хотя в каждом `catch`-блоке объект исключения обозначен одинаково (переменная `e`), речь идет о разных объектах исключений. В некотором смысле инструкция из имени класса ошибки и названия переменной, обозначающей объект исключения, напоминает инструкцию объявления аргумента метода. Переменная, объявленная в `catch`-блоке как объект исключения, доступна только в этом `catch`-блоке.

Генерирование исключений

Для искусственного генерирования исключений используется оператор `throw`, после которого указывается объект исключения. Объект исключения создают специально (как создается любой другой объект), или использую уже сгенерированный объект, переданный для обработки в `catch`-блок

Генерирование исключений

```
class UsingThrowDemo{
   public static void main(String[] args){ // Создание объекта исключения:
      Exception me=new Exception("Искусственная ошибка");
      try{                                 // Контролируемый код (внешний блок)
         System.out.println("Генерируется ошибка");
         try{                              // Контролируемый код (внутренний блок)
            throw me;                      // Генерирование исключения
         } // Обработка исключения (внутренний блок):
         catch(Exception one){
            System.out.println(one);
            System.out.println("А теперь еще одна ошибка");
            throw one;                     // Повторное генерирование исключения
         }
      } // Обработка исключения (внешний блок):
      catch(Exception two){
         System.out.println(two);
      }
      System.out.println("Выполнение программы завершено");
   }
}
```

Генерирование исключений

Генерирование исключений - результат программы

```
Генерируется ошибка
java.lang.Exception: Искусственная ошибка
А теперь еще одна ошибка
java.lang.Exception: Искусственная ошибка
Выполнение программы завершено
```

Контролируемые и неконтролируемые исключения

- Все исключения в Java делятся на контролируемые и неконтролируемые. К неконтролируемым исключениям относятся исключения классов, являющихся подклассами класса `RuntimeException` или класса `Error`. Все остальные исключения относятся к контролируемым

- Разница между исключениями разных типов состоит в том, что для контролируемых исключений автоматически выполняется проверка на наличие обработки исключения: если метод может сгенерировать в процессе выполнения контролируемое исключение, то для такого исключения в методе должна быть предусмотрена обработка

- Если метод все же не содержит код для обработки контролируемого исключения (но потенциально может такое исключение сгенерировать), то в описании метода в явном виде указывается, что он может генерировать необрабатываемое контролируемое исключение: в описании метода после его имени и списка аргументов (но перед фигурными скобками с кодом метода) указывается ключевое слово `throws` и через запятую перечисляются классы контролируемых исключений, которые метод может сгенерировать, но которые в теле метода не обрабатываются

Контролируемые исключения

Класс исключения	Описание
ClassNotFoundException	Ошибка, связанная с невозможностью получения доступа к классу
IllegalAccessException	Ошибка при попытке доступа к ресурсу
InstantiationException	Ошибка с невозможностью создания объекта (например, на основе абстрактного класса)
InterruptedException	Ошибка, связанная с прерыванием одного потока другим потоком
NoSuchFieldException	Ошибка, связанная с отсутствием поля
NoSuchMethodException	Ошибка, связанная с отсутствием метода

На заметку

Классы `ClassNotFoundException`, `IllegalAccessException`, `InstantiationException`, `NoSuchFieldException` и `NoSuchMethodException` являются подклассами класса `ReflectiveOperationException`, который, в свою очередь, является подклассом класса `Exception`. Класс `InterruptedException` является подклассом класса `Exception`. Все перечисленные классы находятся в пакете `java.lang`

Контролируемые исключения

Контролируемые исключения - 1

```
class UsingCheckedExceptionsDemo{
   // Метод выбрасывает контролируемое исключение:
   static void alpha(int n) throws Exception{
      // Текст для передачи аргументом конструктору
      // при создании объекта исключения:
      String txt="Метод alpha() с аргументом "+n;
      // Генерирование исключения:
      throw new Exception(txt);
   }
   // Метод выбрасывает неконтролируемое исключение:
   static void bravo(int n){
      // Текст для передачи аргументом конструктору
      // при создании объекта исключения:
      String txt="Метод bravo() с аргументом "+n;
      // Контролируемый код:
      try{
         if(n>0){
            // Генерирование контролируемого исключения:
            throw new Exception(txt);
         }
         else{
            // Генерирование неконтролируемого исключения:
            throw new RuntimeException(txt);
         }
      }
      // Продолжение на следующем слайде!!!
```

Контролируемые исключения

Контролируемые исключения - 2

```
    // Обработка неконтролируемого исключения:
    catch(RuntimeException e){
       // Повторное генерирование неконтролируемого
       // исключения:
       throw e;
    } // Обработка контролируемого исключения:
    catch(Exception e){
       System.out.println("Обработка ошибки в bravo():");
       System.out.println(e.getMessage());
       System.out.println("****************************");
    }
} // Метод для вызова при обработке исключений.
// Аргументом методу передается объект исключения:
static void catchMe(Exception e){
    System.out.println("Обработка ошибки в main():");
    System.out.println(e.getMessage());
    System.out.println("-------------------------");
} // Главный метод:
public static void main(String[] args){
    // Контролируемый код:
    try{
       // При вызове метода выбрасывается
       // контролируемое исключение класса Exception:
       alpha(123);
    }
    // Продолжение на следующем слайде!!!
```

Контролируемые исключения - 3

```
        // Обработка исключения:
        catch(Exception e){
            // Вызов метода для обработки исключения:
            catchMe(e);
        } // Контролируемый код:
        try{
            // При вызове метода не выбрасывается
            // исключение:
            bravo(123);
        } // Обработка исключения (блок не используется,
        // поскольку исключение не генерируется):
        catch(Exception e){
            // Вызов метода для обработки исключения:
            catchMe(e);
        } // Контролируемый код:
        try{
            // При вызове метода выбрасывается неконтролируемое исключение
            // класса RuntimeException:
            bravo(-1);
        } // Обработка исключения:
        catch(Exception e){
            // Вызов метода для обработки исключения:
            catchMe(e);
        }
    }
}
```

Контролируемые исключения

Контролируемые исключения - результат выполнения программы

```
Обработка ошибки в main():
Метод alpha() с аргументом 123
-------------------------
Обработка ошибки в bravo():
Метод bravo() с аргументом 123
*************************
Обработка ошибки в main():
Метод bravo() с аргументом -1
-------------------------
```

Контролируемые исключения

На заметку

В методе `bravo()` в зависимости от значения аргумента генерируется исключение класса `Exception` или исключение класса `RuntimeException`. Соответственно, в методе предусмотрены `catch`-блоки для обработки исключений данных классов. Специфика ситуации в том, что класс `RuntimeException` является подклассом класса `Exception`. Поэтому `catch`-блок, описанный для исключения класса `Exception`, мог бы обрабатывать и исключения `RuntimeException` если бы не был предусмотрен `catch`-блок специально для обработки исключений класса `RuntimeException`.

При генерировании исключения класса `RuntimeException` оно повторно генерируется в `catch`-блоке, причем с тем же объектом исключения, что и первый раз. В результате метод `bravo()` при неположительных значениях аргумента выбрасывает исключение класса `RuntimeException`. Но если мы попытаемся добиться того же эффекта, ограничившись однократным генерированием исключения класса `RuntimeException`, отказавшись от обработки (и повторного генерирования) этого исключения в `catch`-блоке, то такое исключение будет обработано в `catch`-блоке, предназначенном для обработки исключений класса `Exception`. И все потому, что класс `RuntimeException` является подклассом класса `Exception`

Пользовательские классы исключений

- Существует возможность создавать пользовательские классы для исключений: на основе одного из классов исключений путем наследования создается класс

- Объект, созданный на основе такого класса, может использоваться при генерировании исключительных ситуаций

- Например, в качестве суперкласса можем использовать класс `Exception`, или, скажем, класс `RuntimeException`

- В первом случае получим пользовательский класс исключения контролируемого типа, а во втором - неконтролируемого типа

Пользовательские классы исключений

Пользовательские исключения - 1

```
// Класс контролируемого исключения создается наследованием класса Exception:
class MyException extends Exception{
   private int code;     // Закрытое числовое поле
   MyException(int n){   // Конструктор
      super();
      code=n;
   } // Переопределение метода toString():
   public String toString(){
      String txt="Исключение класса MyException\n";
      txt+="Код ошибки: "+code+"\n";
      txt+="-----------------------------";
      return txt;
}} // Класс неконтролируемого исключения создается
// наследованием класса RuntimeException:
class MyMistake extends RuntimeException{
   private int code; // Закрытое числовое поле
   MyMistake(int n){ // Конструктор
      super();
      code=n;
   } // Переопределение метода toString():
   public String toString(){
      String txt="Исключение класса MyMistake\n";
      txt+="Код ошибки: "+code+"\n";
      txt+="*****************************";
      return txt;
}} // Продолжение на следующем слайде!!!
```

Пользовательские исключения - 2

```
// Главный класс:
class UsingMyExceptionDemo{
   // Статический метод выбрасывает контролируемое
   // исключение класса MyException:
   static void alpha(int n) throws MyException{
      throw new MyException(n);
   } // Статический метод выбрасывает неконтролируемое
   // исключение класса MyMistake:
   static void bravo(int n){
      throw new MyMistake(n);
   } // Главный метод:
   public static void main(String[] args){
      try{                          // Контролируемый код (внешний блок)
         try{                       // Контролируемый код (внутренний блок)
            alpha(123);             // Метод выбрасывает исключение MyException
         } // Обработка исключения класса MyException:
         catch(MyException e){
            System.out.println(e);
            bravo(321);
         }
      } // Обработка исключения класса MyMistake:
      catch(MyMistake e){
         System.out.println(e);
      }
   }
}
```

Пользовательские классы исключений

Пользовательские исключения - результат программы

```
Исключение класса MyException
Код ошибки: 123
------------------------------
Исключение класса MyMistake
Код ошибки: 321
******************************
```

Домашнее задание

- Проанализировать программный код примеров, представленных в презентации
- Набрать код и проверить его функциональность

Продолжение следует...