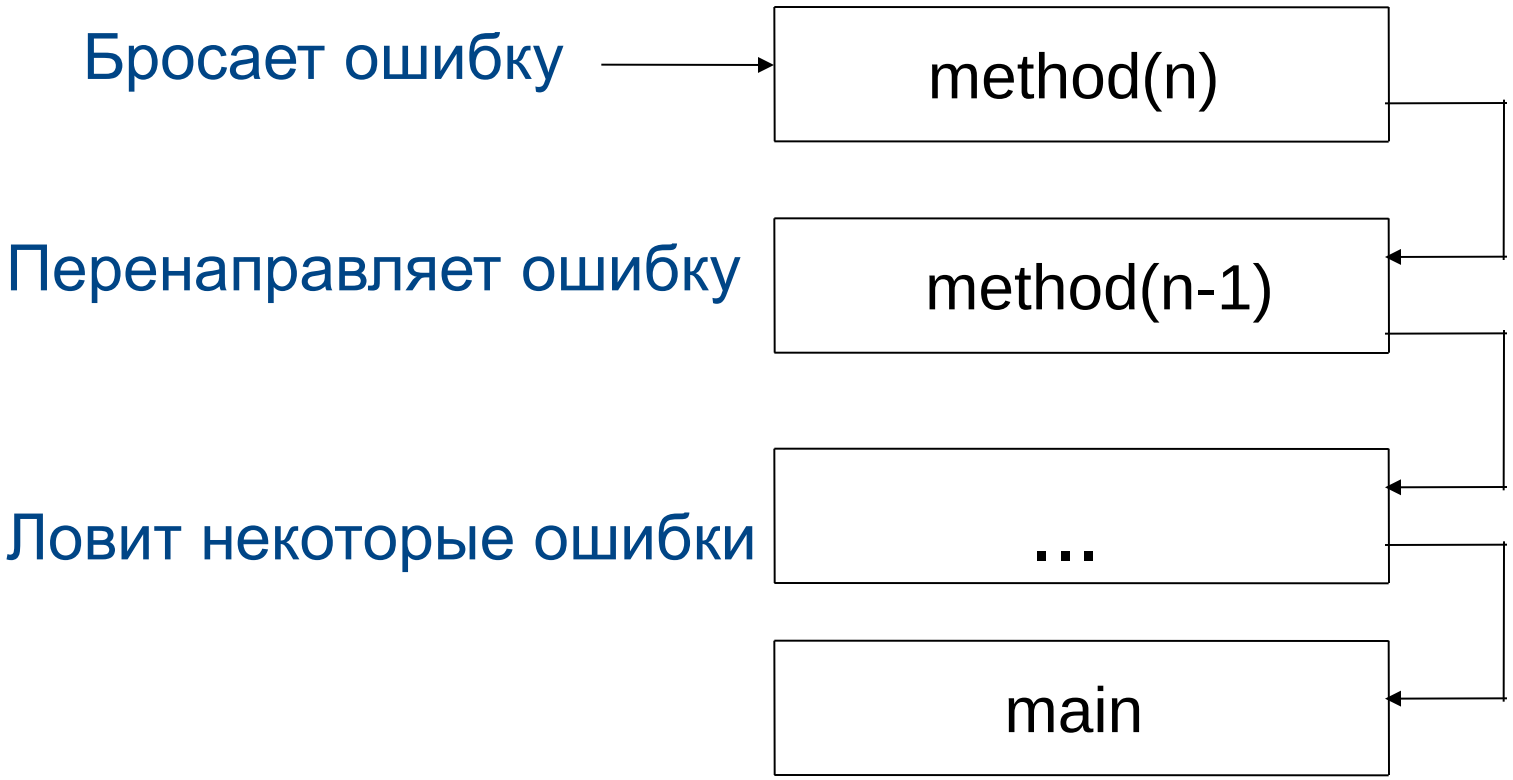
Модуль #8
Исключение в Java

Исключения

- **Общая информация**
- Декларация исключений
- Типы исключений
- Блок finally
- Возбуждение исключений
- Собственные классы исключений
- Исключения и замещение метода

Исключение (Exception) – событие возникающее в процессе работы программы и прерывающее ее нормальное исполнение.



Примеры:

- `java.lang.NullPointerException`
- `java.lang.IllegalStateExcpetion`
- `java.lang.ArrayIndexOutOfBoundsException`
- `java.lang.ClassCastException`
- `java.lang.OutOfMemoryError`
- `java.io.IOException`
- `java.lang.ArithmeticException`
- `java.lang.StackOverflowException`

- **Исключение** – это полноценный объект в java.
- Все исключения наследуются от класса Throwable.

Доступные методы из класса Throwable:

- `String getMessage()`
- `Throwable getClause()`
- `StackTraceElement[] getStackTrace()`
- `void printStackTrace()`

Как читать StackTrace?

```
Exception in thread "Main”
java.lang.NullPointerException
        at ua.luxoft.java.Test.baz(Test.java:36)
        at ua.luxoft.java.Test.bar(Test.java:240)
        at ua.luxoft.java.Test.foo(Test.java:120)
        at ua.luxoft.java.Test.main(Test.java:360)
```

main, foo, bar, baz – названия методов.

36, 240, 120, 360 – строки в которых произошел вызов метода

Перехват исключений

Для перехвата исключений можно воспользоваться блоком `try/catch`.

```
try
{
    // Блок кода который может бросить исключение
    /*
     *  Любой другой код
     */
}
catch (Exception ex)
{
    // Реакция на исключительную ситуацию
}
```

Перехват исключений:

```
try
{
    // Блок кода, бросающий исключение (тип 1)
    // Любой код
    // Блок кода, бросающий исключение (тип 2)
}
catch (FileNotFoundException ex)
{
    // Реакция на исключительную ситуацию (тип 1)
}
catch (ParseException ex)
{
    // Реакция на исключительную ситуацию (тип 2)
}
```

Перехват исключений:

В Java 7 появилась конструкция multi-catch.

```
try
{
   // Блок кода, бросающий исключение (IOException)
   // Любой код
   // Блок кода, бросающий исключение (ParseException)
}
catch (IOException | ParseException ex)
{
   // Реакция на исключительную ситуацию (типы 1 и 2)
}
```

Особенности перехвата исключений:

- Каждый блок `catch` работает аналогично оператору `instanceof`.
- Если `catch` ловит IOException, то он поймает также всех его наследников.
- Блоки `catch` должны идти в таком порядке, чтобы тип исключений нижеследующих блоков был выше по иерархии наследования.

Особенности перехвата исключений:

- Компилятор отслеживает неправильное расположение блоков `catch`, и сообщает о том, что `catch` блок никогда не выполнится.

- Если код блока `try` не возбудил исключение, то вызывается код, следующий сразу после последнего блока `catch`.

- Блоки `catch` должны идти в таком порядке, чтобы тип исключений нижеследующих блоков был выше по иерархии наследования.

Пример:

```
File myFile = getMyFile();
try
{
   myFile.isDirectory();
}
catch (IOException exception)
{
   exception.printStackTrace();
}
System.out.println("Life goes on");
```

Пример:

```
int[] m = {-1,0,1};
Scanner sc = new Scanner(System.in);
try
{
   int a = sc.nextInt();
   m[a-1] = 4/a;
   System.out.println(m[a]);
}
catch (ArithmeticException e)
{
    System.out.println("Недопустимая операция");
}
catch (ArrayIndexOutOfBoundsException e)
{
   System.out.println("Недопустимый индекс массива");
}
catch (Exception e)
{
   System.out.println("Ещё какое-то исключение");
}
```

Исключения

- Введение
- **Декларация исключений**
- Типы исключений
- Блок finally
- Возбуждение исключений
- Собственные классы исключений
- Исключения и замещение метода

Исключения без `catch` блока

```
private static int throwsTwo() throws
                    IOException, AWTException
{
    /*
     *   Любой другой код
     */
     // Метод бросающий исключение
}
```

- Перехват или Обработка?

Перехват или Обработка?

- **Главное правило** – если метод обладает информацией о том, как правильно обработать исключение, он может его обработать, иначе он должен делегировать.

Перехват или Обработка?

Существует также «смешанная» стратегия обработки исключения:

```
public void f() throws MyException {
    try {
        aMethod();
    } catch (MyException e) {
        System.out.println("Exception was thrown while
calling method aMethod()");
        throw e;
    }
}

private void aMethod() throws MyException{
    if (...)
        throw new MyException();
}
```

Исключения

- Введение
- Декларация исключений
- **Типы исключений**
- Блок finally
- Возбуждение исключений
- Собственные классы исключений
- Исключения и замещение метода

Типы исключений

В Java есть два типа исключений:

- Контролируемые (checked)
- Некотролируемые (unchecked)

Пример:

Выход за пределы массива и обращение к несуществующему файлу.

Типы исключений

Runtime исключение – ошибка программиста, возникающая при дизайне, отладке и разработке кода.

- Не бросать runtime исключения в завершенном коде
- Runtime исключения можно не обрабатывать
- Имеют java.lang.RuntimeException как базовый класс

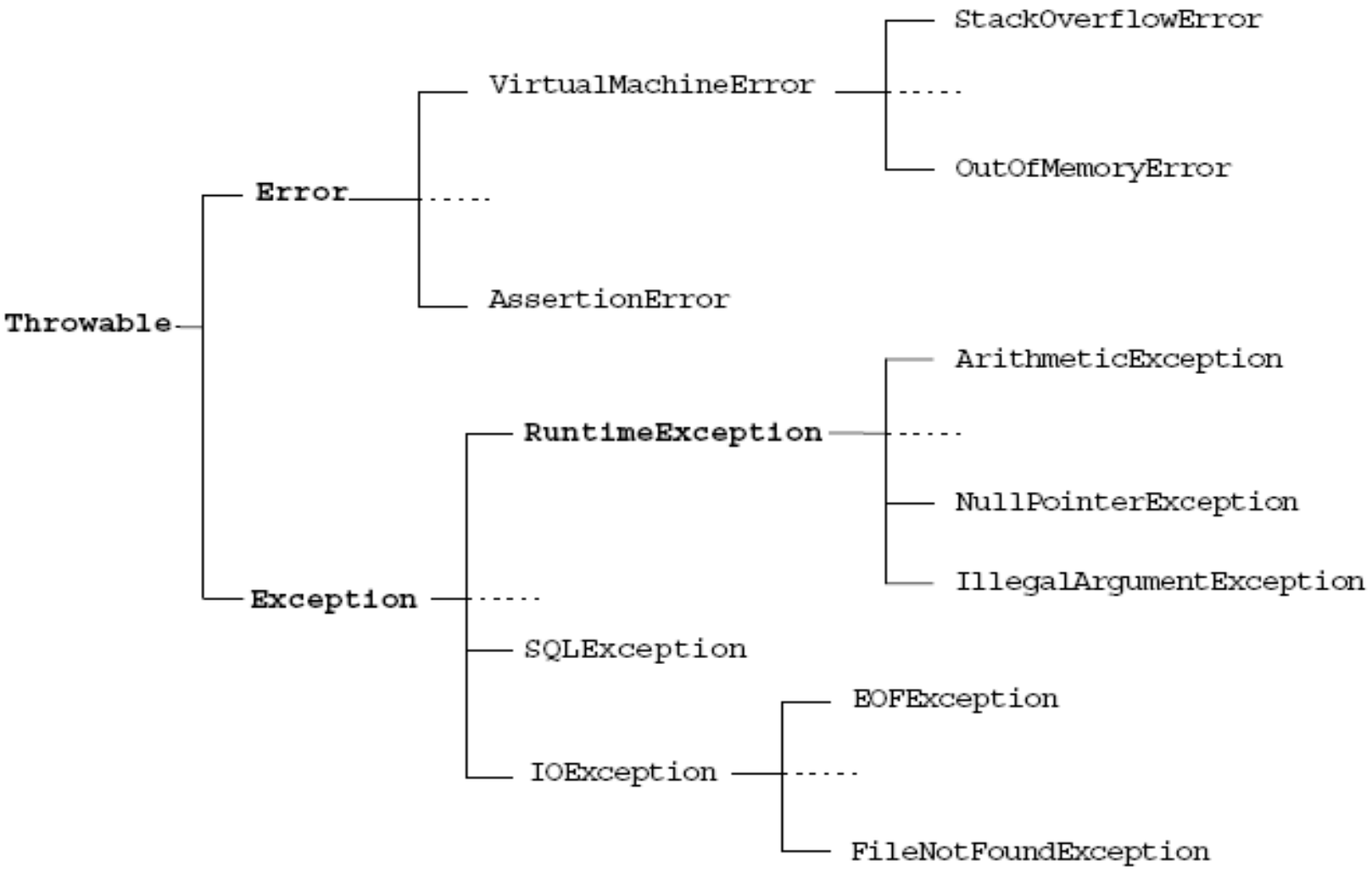
Внимание! Все прочие наследники **`java.lang.Exception`** являются checked исключениями. Компилятор проверяет, что для каждого checked исключения где-либо существует **`catch`** обработчик

Типы исключений

- Error – ошибки, возникающие в устройстве и реализации виртуальной машины.

- Программист не должен ловить или возбуждать ошибки, они зарезервированы для JVM.

Иерархия исключений



Исключения

- Введение
- Декларация исключений
- Типы исключений
- **Блок finally**
- Возбуждение исключений
- Собственные классы исключений
- Исключения и замещение метода

Блок finally

После последнего блока **`catch`** может идти блок **`finally`**. Гарантируется, что код этого блока будет выполнен *почти всегда*. А именно:

- Если **`try`** или **`catch`** блоки выполнились успешно.
- Если они в свою очередь так же возбудили. исключение.
- Если они произвели выход из метода через **return**

Внимание! Блок finally не будет вызван если:

1. Поток завершен вызовом interrapt.
2. Принудительное завершение программы (`System.exit(0)`);

Блок finally

```
// Любой код
try
{
    // Блок кода, бросающий исключение
    // Любой код
}
catch (Exception ex)
{
    // Реакция на исключительную ситуацию
}
finally
{
     // Этот код будет выполенен в 99.999%
}
```

Блок finally

```
File myFile = getMyFile();
String s = getStringAndHopeForAllDigists();
try
{
    System.out.println("About to call");
    int x = Integer.parseInt(s);
    myFile.getCanonicalFile();
    System.out.println("Call succeded");
}
catch (NumberFormatException ex)
{
    System.out.println("Bad text" + ex.getMessage());
}
catch(IOException ex)
{
    System.out.println("File stress" + ex.getMessage());
}
finally
{
    System.out.println("In the block finally");
}
```

Блок finally

Что выведется на консоль?

```
try
{
   try
   {
      throw new IllegalStateException();
   }
   finally
   {
      throw new RuntimeException();
   }
}
catch (IllegalStateException ex)
{
   System.out.println(“1”);
}
catch (RuntimeException e)
{
   System.out.println(“2”);
}
catch (Exception ee)
{
   System.out.println(“3”);
}
```

Исключения

- Введение
- Декларация исключений
- Типы исключений
- Блок finally
- **Возбуждение исключений**
- Собственные классы исключений
- Исключения и замещение метода

Возбуждение исключений

- Выбрать соответствующее ситуации исключение.
- Для возбждения исключительной ситуации используется ключевое слово **`throw`**.
- Обычно в конструктор исключения передают строку, описывающую ситуацию более детально.

```
throw new IOException("Connection closed")
```

Внимание! StackTrace создается с конекстом того места где создано исключение.

Возбуждение исключений

- Компилятор следит за тем, чтобы возбуждаемое исключение было декларировано в списке `throws`
- Можно декларировать базовый класс.
- Если метод возбуждает несколько checked исключений, то все они, либо их родитель, должны быть указаны в `throws`.

Исключения

- Введение
- Декларация исключений
- Типы исключений
- Блок finally
- Возбуждение исключений
- **Собственные классы исключений**
- Исключения и замещение метода

Собственные классы исключений

- Можно создавать собственные классы исключений.
- Контролируемые исключения должны расширять `java.lang.Exception` или один из его подклассов.
- Runtime исключения должны расширять `java.lang.RuntimeException`
- Собственное исключение не должно иметь никаких методов и данных, однако, может их указывать для того, чтобы лучшим образом описать исключительную ситуацию.

Внимание! Перед созданием собственного исключения стоит проверить наличие существующего.

Исключения

- Введение
- Декларация исключений
- Типы исключений
- Блок finally
- Возбуждение исключений
- Собственные классы исключений
- **Исключения и замещение метода**

Исключение и замещение метода

- При замещении метода в подклассе компилятор проверяет, что все классы исключений, выбрасываемые этим методом, те же или подклассы этих исключений.

```
public class TestA {
    public void methodA() throws IOException {
some file manipulation
    }
}
public class TestB1 extends TestA {
    public void methodA() throws EOFException {
        // do some file manipulation
    }
}
public class TestB2 extends TestA {
    public void methodA() throws Exception { //
        // do some file manipulation
    }
}
```

Что делать нельзя!

- Строить логику работы приложения на исключениях.

```
try
{
    int index = 0;
    while(true)
    {
        System.out.println(array[i++]);
    }
}
catch (IndexOutOfBoundsException e)
{
    // Все хорошо, конец массива
}
```

Что делать нельзя!

- Нельзя игнорировать исключения. Никто никогда не узнает что было брошено исключения.

```
try
{
    // Бросается ошибка
}
catch (IOException e)
{

}
```

Что делать нельзя!

- Запись в лог или консоль.

```
try
{
    // Бросается ошибка
}
catch (IOException e)
{
    e.printStackTrace();
}
```

Что делать нельзя!

- Не нужно ловить все подряд.

```
try
{
    // Бросается ошибка
}
catch (Exception e)
{

}
```

Что делать нельзя!

- Перебразывание исключений

```
try
{
    // Бросается ошибка
}
catch (IOException e)
{
    throw new AccountException(e);
}
```

правильно

```
try
{
    // Бросается ошибка
}
catch (IOException e)
{
    e.printStackTrace();
    throw new AccountException();
}
```

не правильно

Упражнение 11

Bank Application