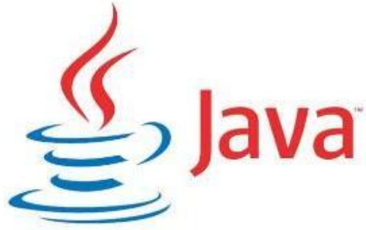
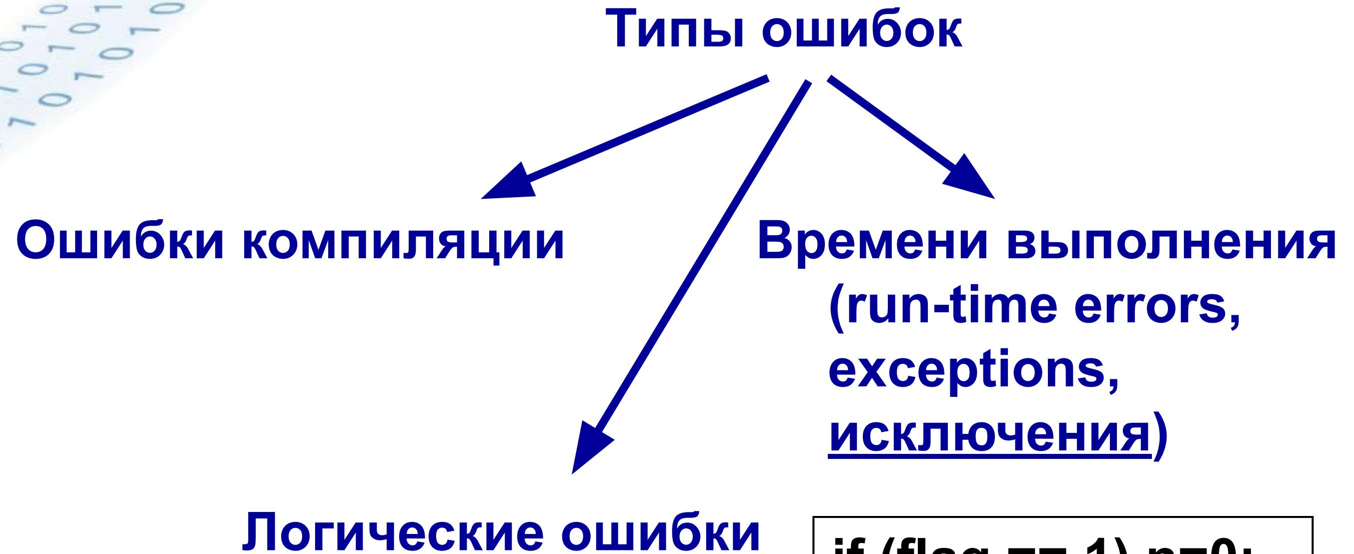
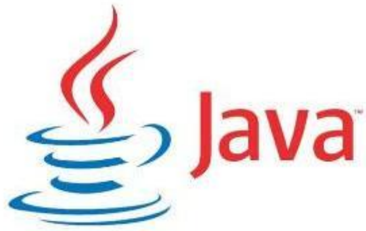


Обработка исключений

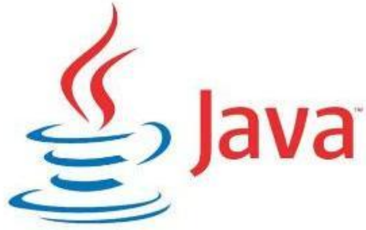



```
if (flag == 1) n=0;
else n=1;
val=1/n;
```



Обработка исключений

Исключение в Java — это объект некоторого класса, который описывает исключительное состояние, возникшее в каком-либо участке программного кода. При возникновении исключения исполняющая система Java создает объект класса, связанного с данным исключением. Этот объект хранит информацию о возникшей исключительной ситуации (точка возникновения, описание и т.п.) Возможна как автоматическая так и программная генерация исключений.



Обработка исключений

Формат try-catch блока

try

{ // блок кода }

catch (ExceptionType1 exOb1)

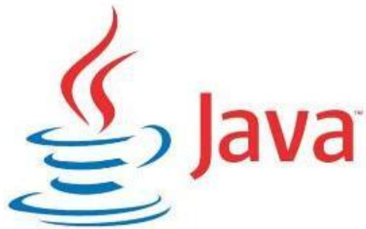
{ // обработчик исключений типа ExceptionType1 }

[catch (ExceptionType2 exOb2)

{ // обработчик исключений типа ExceptionType2 }]

[finally

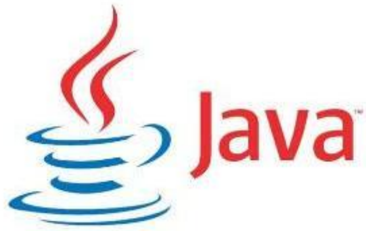
{ //код, который выполняется перед выходом из блока try }]



Обработка исключений

Формат try-catch блока

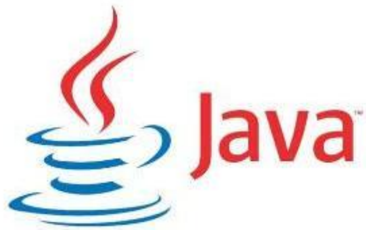
```
try
{ read_from_file ("data.txt");
  calculate();
}
catch ( FileNotFoundException fe )
{  System.out.println("Файл data.txt не найден");
}
catch ( ArithmeticException aex )
{ System.out.println("Деление на ноль");
}
```



Обработка исключений

Множественный перехват исключений

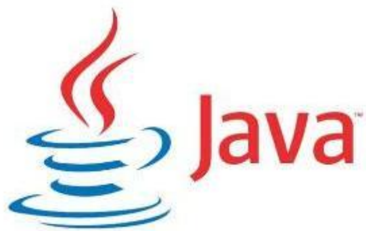
```
try {
  read_from_file ("data.txt");
  calculate();
}
catch ( FileNotFoundException | ArithmeticException aex ) {
   System.out.println(“Something is wrong");
}
```



Обработка исключений

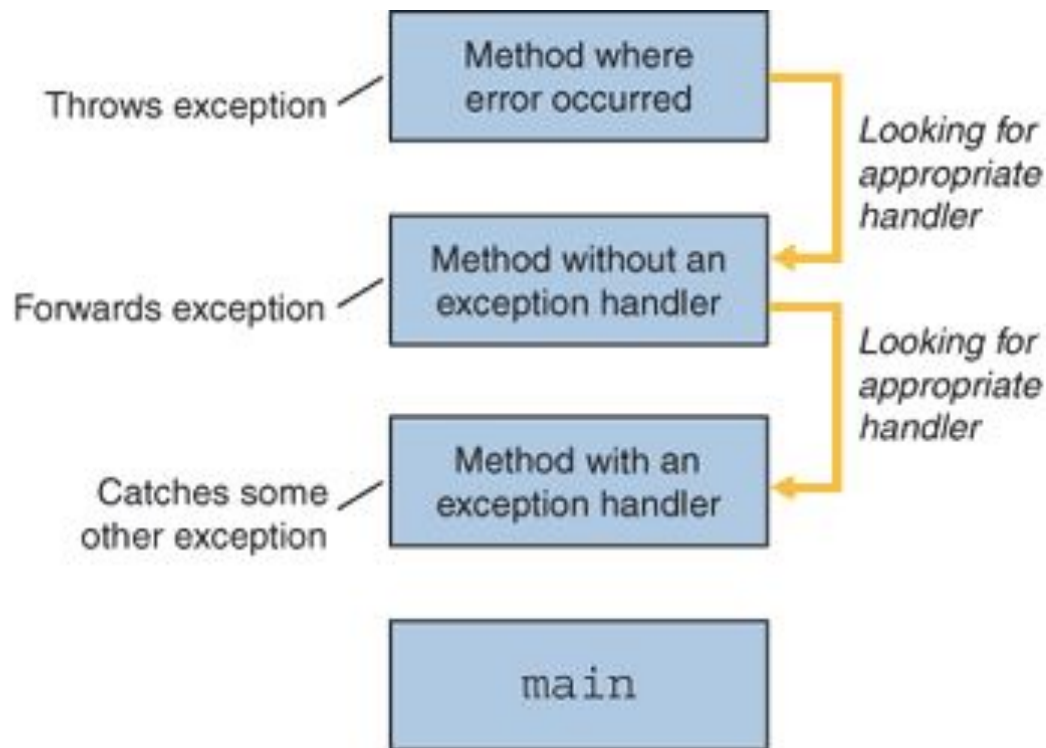
Формат try-catch блока

- управление никогда не возвращается из блока catch обратно в блок try, после выполнения catch-блока управление передается строке, следующей сразу после try-catch-блока;
- область видимости catch-блока ограничена ближайшим предшествующим утверждением try, т.е. catch-блок не может захватывать исключение, выброшенное «не своим» try-блоком;
- операторы, контролируемые утверждением try, должны быть окружены фигурными скобками даже если это одиночная инструкция.
- блоки try могут быть вложенными

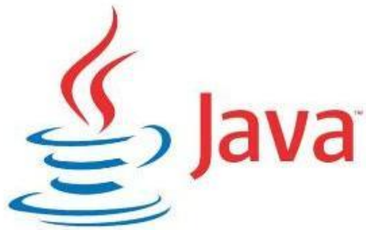


Обработка исключений

Стек вызовов



Если обработчик не найден исключение передается JVM.

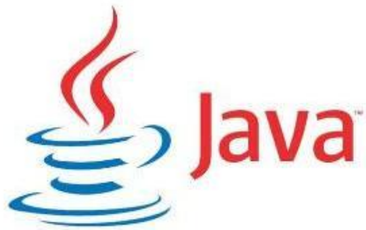


Обработка исключений
Дефолтный обработчик исключений

cmd - Shortcut

```
C:\temp\jad>java Broken
Exception in thread "main" java.lang.NullPointerException
        at Broken.doEvenMoreStuff(Broken.java:22)
        at Broken.doMoreStuff(Broken.java:16)
        at Broken.doStuff(Broken.java:11)
        at Broken.main(Broken.java:36)

C:\temp\jad>
```



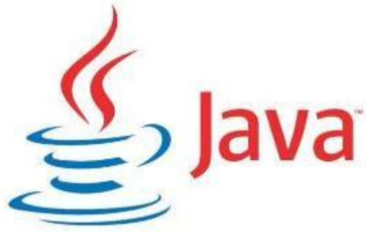
Обработка исключений
Классы исключений

Object
Throwable
Error
Exception
RuntimeException

никогда не обрабатываются, т.к. не имеет смысла

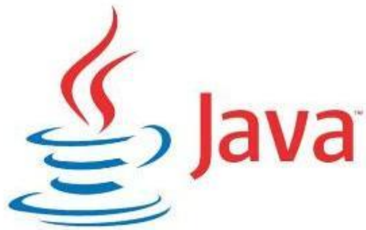
обязательно обрабатывать, иначе код не скомпилируется

надо обрабатывать, но компилятор это не проверяет



Обработка исключений
Классы исключений

Catch-блоки просматриваются в порядке их появления в программе, при этом обработчик catch для суперкласса перехватывает исключения как для своего класса так и для всех его подклассов. Следовательно, в последовательности catch-блоков подклассы исключений должны следовать перед любым из суперклассов.



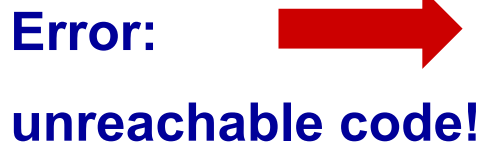
Обработка исключений

Классы исключений

```
try
{inputFile("data.txt");
 calculate();  }
```

```
catch ( FileNotFoundException ExObj1)
catch ( IOException ExObj2)
{System.out.println("Файл data.txt не найден");} }
{System.out.println("Ошибка ввода-вывода");} }
```

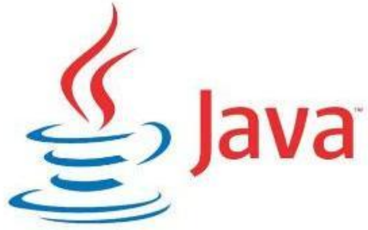
```
catch ( FileNotFoundException ExObj1)
catch ( IOException ExObj2)
{System.out.println("Файл data.txt не найден");} }
{System.out.println("Ошибка ввода-вывода");} }
```



Error:

unreachable code!

т.к. FileNotFoundException – подкласс IOException

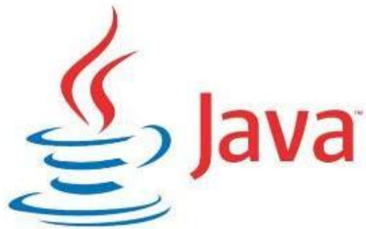


Обработка исключений

Оператор throw

Программная генерация исключения:

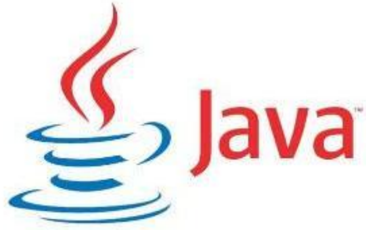
```
throw  new <ExceptionClassName>();
throw  new <ExceptionClassName>("…");
```



Обработка исключений

Оператор throw

```
public void demoproc ()
{try
  {throw new NullPointerException("demo"); }
 catch (NullPointerException e)
   {System.out.println("caught inside demoproc"); throw e; }
}
public static void main(String args[])
{try
  {demoproc(); }
catch(NullPointerException e)
  {System.out.println("recaught: " + e); }
}
```

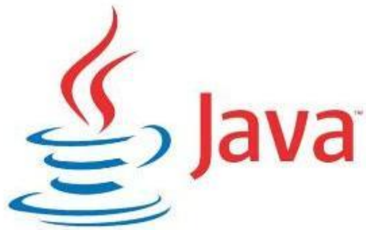


Обработка исключений

Оператор throw

Вывод программы:

```
C:\Program Files\Xinox Software\JCreatorV3\GE2001.exe
caught inside demoproc
recaught: java.lang.NullPointerException: demo
Press any key to continue...
```



Обработка исключений

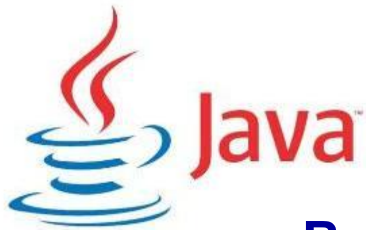
Оператор throws

Исключения, которые порождены от Exception, но не от RuntimeException, могут быть сгенерированы только явно операцией throw. При этом если метод может выбрасывать одно из таких исключений, то должно выполняться одно из двух условий: либо для такого исключения должен быть catch-обработчик, либо в заголовке такого метода должна стоять конструкция:

throws <ExceptionClassName1> [,<ExceptionClassName2>,…]

```
public String readLine() throws IOException
```

(т.н. *Catch or Specify Requirement*)



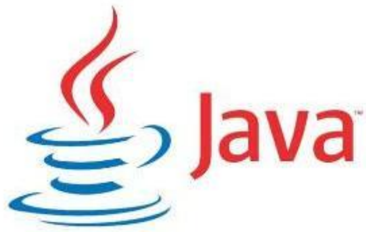
Обработка исключений

Оператор throws

Вызов метода, в описании которого стоит " throws ... ", должен находиться либо внутри try-catch-блока, либо также внутри метода с конструкцией " throws ... " в его заголовке и т.д. вплоть до метода main().

Таким образом, где-то в программе любое возможное исключение, относящееся к категории «checked», обязано быть перехвачено и у компилятора есть возможность это проконтролировать.

Большинство библиотечных методов определено с ключевым словом throws, специально чтобы оставить конкретную обработку исключения на усмотрение вашего приложения. Вызов таких методов приводит к необходимости учитывать Catch or Specify requirement.

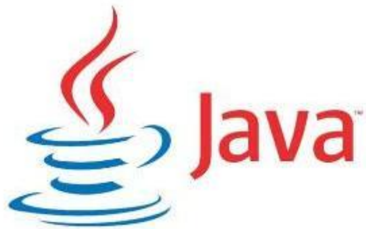


Обработка исключений

Подклассы Exception

```
class MyException extends Exception
{private int detail;
 MyException(int a)
    {detail = a; }
 public String toString()
    {return "MyException[" + detail + "]"; }
}
```

В данном случае создано checked-exception. Если надо создать unchecked-exception, то его надо унаследовать от RuntimeException.



Обработка исключений

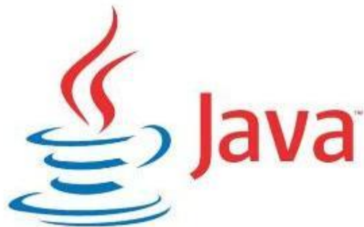
Try-with-resources

В Java SE 6:

```
static String readFirstLineFromFile(String path) throws IOException {
  BufferedReader br = new BufferedReader(new FileReader(path));
  try {
    return br.readLine();
  } finally {
    if (br != null) br.close();
  }
}
```

Неудобство: надо вручную закрывать ресурсы в блоке finally

Проблема: Если оба метода readLine() и close() выбросят исключения, то из метода будет выброшено исключение, которое возникло в блоке finally, а то которое возникло в блоке try будет перекрыто.



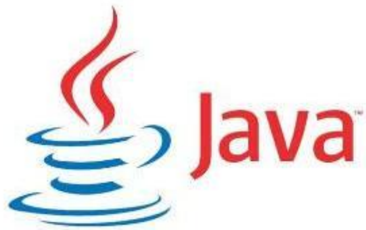
Обработка исключений
Try-with-resources

В JavaSE 7 введен новый интерфейс java.lang.AutoCloseable. Объекты, реализующие AutoCloseable, если они созданы в try … catch блоке, освобождаются (закрываются) автоматически при выходе из блока вне зависимости от того, возникло ли исключение.

```
static String readFirstLineFromFile(String path) throws IOException {
  try (BufferedReader br = new BufferedReader(new FileReader(path))) {
    return br.readLine();
  }
```

try-with resource statement

Если исключения выбрасываются из try блока и из try-with-resources, то из метода выбрасывается исключение, которое было выброшено в try; исключение выброшенное из try-with-resources подавляется (suppressed) Оно может быть получено с помощью метода Throwable.getSuppressed()



Обработка исключений
Try-with-resources

```
try (
    java.util.zip.ZipFile zf = new java.util.zip.ZipFile(zipFileName);
    java.io.BufferedWriter writer =
java.nio.file.Files.newBufferedWriter(outputFilePath, charset)
   )
{…}
```

Try-with-resources оператор может содержать несколько объявлений, разделенных “;” При выходе из блока (нормальном или с исключением) автоматически вызываются методы close() для объектов BufferedWriter и ZipFile в порядке, обратном созданию. Блоки catch и finally если они есть всегда отрабатывают после освобождения ресурсов.