Оглавление

[1 Установка java 2](#_Toc490128755)

[2 Вложенные классы 2](#_Toc490128756)

[2.1 Обычный вложенный класс 2](#_Toc490128757)

[2.2 Локальный класс (вложенный в метод)– определен в каком-либо методе 3](#_Toc490128758)

[2.3 Анонимный вложенный класс 3](#_Toc490128759)

[2.4 Статический вложенный класс 3](#_Toc490128760)

[3 Swing интерфейс (JComponent для рисования и JPannel для элементов): 4](#_Toc490128761)

[4 Обработка событий 5](#_Toc490128762)

[4.1 Создать отдельный класс, имплементирующий интерфейс 5](#_Toc490128763)

[4.2 Создать анонимный вложенный класс 5](#_Toc490128764)

[4.3 Использовать классы – адаптеры: создать класс, расширяющий класс-адаптер, или создать сам класс - адаптер. (Здесь создается анонимный класс адаптер) 6](#_Toc490128765)

[4.4 Использовать интерфейс Action (действие) и реализующий его класс AbstractAction. Это для того, чтобы обрабатывать одну и туже команду из разных источников событий (меню, нажатие клавиш и т.д.). 7](#_Toc490128766)

[5 Сборка jar и размещение на сервере Tomcat (java Web Start) 8](#_Toc490128767)

[5.1 Создание jar 8](#_Toc490128768)

[5.2 Размещение на сервере tomcat (java web start) 10](#_Toc490128769)

[5.2.1 Подготовить файл jnlp 10](#_Toc490128770)

[1.1.1 Установить Tomcat: 10](#_Toc490128771)

[1.1.1 Поместить файлы в каталог: 11](#_Toc490128772)

[6 Applet 13](#_Toc490128773)

[7 Создать свой класс исключения: 13](#_Toc490128774)

[8 Диагностические утверждения 14](#_Toc490128775)

[9 Логирование 14](#_Toc490128776)

[10 StackTrace 15](#_Toc490128777)

[11 Обобщенные классы (generic) 15](#_Toc490128778)

[12 Рефлексия 17](#_Toc490128779)

[13 Потоки 18](#_Toc490128780)

[13.1 Синхронизация потоков 18](#_Toc490128781)

[13.1.1 Метод 1 (более новый): реентерабельная блокировка 18](#_Toc490128782)

[13.1.2 Метод синхронизации 2 (простой): synchronized 19](#_Toc490128783)

[13.1.3 Синхронизированные блоки 19](#_Toc490128784)

[13.1.4 Volatile – обеспечивает атомарность поля 20](#_Toc490128785)

[13.1.5 Не рекоментованные методы(устаревшие): 20](#_Toc490128786)

[13.2 Пул потоков 20](#_Toc490128787)

[**14** **Потоки и файлы** 22](#_Toc490128788)

[14.1 Текстовые данные. 23](#_Toc490128789)

[14.1.1 Запись текстовых данных. 23](#_Toc490128790)

[**14.1.2** **Считывание текстовых данных.** 24](#_Toc490128791)

[14.2 **Двоичные данные.** 25](#_Toc490128792)

[**14.2.1** **Произвольный доступ к файлам –** можно как читать, так и записывать(это указывается во втором параметре) 25](#_Toc490128793)

[**14.2.2** **zip – архивы** 26](#_Toc490128794)

[14.3 **Сериализация.** 27](#_Toc490128795)

[14.4 **Отображаемые в памяти файлы.** 27](#_Toc490128796)

[14.5 **Регуляные выражения.** 28](#_Toc490128797)

[**15** **XML** 28](#_Toc490128798)

[**16** **Подпись данных** 29](#_Toc490128799)

[**17** **Анонимные методы (лямбда-выражения)** 30](#_Toc490128800)

# Установка java

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Переменные среды:

|  |
| --- |
| JAVA\_HOME |
| C:\Program Files\Java\jdk1.8.0\_141 |

|  |
| --- |
| Path |
| C:\Program Files\Java\jdk1.8.0\_141\bin; |

Это для доступа к различным библиотекам при компиляции из командной строки:

|  |
| --- |
| CLASSPATH |
| C:\Program Files\Java\jdk1.8.0\_141\jre\lib\; |

Проверка:

java – version

Документация по API:

http://docs.oracle.com/javase/8/docs/

http://docs.oracle.com/javase/8/docs/api/

https://help.javarush.ru/

# Вложенные классы

## Обычный вложенный класс

|  |
| --- |
| **class** TalkingClock {  **public class** TimePrinter **implements** ActionListener {  **public void** actionPerformed(ActionEvent event) {  Date now = **new** Date();  System.***out***.println(**"At the tone, the time is "** + now);  **if** (beep) Toolkit.*getDefaultToolkit*().beep();  }  } } |

## Локальный класс (вложенный в метод)– определен в каком-либо методе

|  |
| --- |
| **class** TalkingClock {  **public void** start() {  **class** TimePrinter **implements** ActionListener {  **public void** actionPerformed(ActionEvent event) {  Date now = **new** Date();  System.***out***.println(**"At the tone, the time is "** + now);  **if** (**beep**) Toolkit.*getDefaultToolkit*().beep();  }  } }  } |

## Анонимный вложенный класс

|  |
| --- |
| **class** TalkingClock {  **public void** start(**int** interval, **final boolean** beep) {  ActionListener listener = **new** ActionListener() {  **public void** actionPerformed(ActionEvent event) {  System.***out***.println(**"At the tone, the time is "**);  **if** (beep) Toolkit.*getDefaultToolkit*().beep();  }  };  Timer t = **new** Timer(interval, listener);  t.start();  } } |

## Статический вложенный класс

|  |
| --- |
| **public class** StaticInnerClassTest {  **public static void** main(String[] args) {  **double**[] d = **new double**[20];  **for** (**int** i = 0; i < d.**length**; i++)  d[i] = 100 \* Math.*random*();  ArrayAlg.Pair p = ArrayAlg.*minmax*(d);  System.***out***.println(**"min = "** + p.getFirst());  System.***out***.println(**"max = "** + p.getSecond());  } }  **class** ArrayAlg {   **public static** Pair minmax(**double**[] values) {  **double** min = Double.***MAX\_VALUE***;  **double** max = Double.***MIN\_VALUE***;  **for** (**double** v : values) {  **if** (min > v) min = v;  **if** (max < v) max = v;  }  **return new** Pair(min, max);  }   **public static class** Pair {   **private double first**;  **private double second**;   **public** Pair(**double** f, **double** s) {  **first** = f;  **second** = s;  }   **public double** getFirst() {  **return first**;  }   **public double** getSecond() {  **return second**;  }  } } |

# Swing интерфейс (JComponent для рисования и JPannel для элементов):

|  |
| --- |
| **public class** Test {  **public static void** main(String[] args) {  EventQueue.*invokeLater*(**new** Runnable() {  **public void** run() {  NotHelloWorldFrame frame = **new** NotHelloWorldFrame();  frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  frame.setVisible(**true**);  }  });  } }  **class** NotHelloWorldFrame **extends** JFrame {  **public** NotHelloWorldFrame() {  setTitle(**"NotHelloWorld"**);  setSize(300, 200);   NotHelloWorldComponent comp = **new** NotHelloWorldComponent();  comp.setSize(300,200);  add(comp);   MyJPanel mb = **new** MyJPanel(comp);  add(mb);  } }  **class** NotHelloWorldComponent **extends** JComponent {   **public void** paintComponent(Graphics g) {  setBackground(Color.***RED***);  g.drawString(**"Hello: "**+Math.*random*(), 75, 100);  } }  **class** MyJPanel **extends** JPanel {  MyJPanel(NotHelloWorldComponent comp) {  JButton b = **new** JButton(**"Print Hello"**);  add(b);  b.addActionListener(**new** ActionListener() {  @Override  **public void** actionPerformed(ActionEvent e) {  comp.repaint();  }  });  } } |

# Обработка событий

## Создать отдельный класс, имплементирующий интерфейс

|  |
| --- |
| **import** javax.swing.\*; **import** java.awt.\*; **import** java.awt.event.ActionEvent; **import** java.awt.event.ActionListener;  **public class** NotHelloWorld {  **public static void** main(String[] args) {  EventQueue.*invokeLater*(**new** Runnable() {  **public void** run() {  NotHelloWorldFrame frame = **new** NotHelloWorldFrame();  frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  frame.setVisible(**true**);  }  });  } }  **class** NotHelloWorldFrame **extends** JFrame {  **public static final int *DEFAULT\_WIDTH*** = 300;  **public static final int *DEFAULT\_HEIGHT*** = 200;  **public** NotHelloWorldFrame() {  setTitle(**"NotHelloWorld"**);  setSize(***DEFAULT\_WIDTH***, ***DEFAULT\_HEIGHT***);    MyJPanel mb = **new** MyJPanel();  add(mb);   } }  **class** MyJPanel **extends** JPanel {  MyJPanel() {  JButton myButton = **new** JButton(**"Hello"**);  ActionListener listener = **new** MyActionListener();  myButton.addActionListener(listener);  add(myButton);  } }  **class** MyActionListener **implements** ActionListener{  @Override  **public void** actionPerformed(ActionEvent e) {  System.***out***.println(**"Обработчик события"**);  } } |

## Создать анонимный вложенный класс

|  |
| --- |
| **import** javax.swing.\*; **import** java.awt.\*; **import** java.awt.event.ActionEvent; **import** java.awt.event.ActionListener;  **public class** NotHelloWorld {  **public static void** main(String[] args) {  EventQueue.*invokeLater*(**new** Runnable() {  **public void** run() {  NotHelloWorldFrame frame = **new** NotHelloWorldFrame();  frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  frame.setVisible(**true**);  }  });  } }  **class** NotHelloWorldFrame **extends** JFrame {  **public static final int *DEFAULT\_WIDTH*** = 300;  **public static final int *DEFAULT\_HEIGHT*** = 200;   **public** NotHelloWorldFrame() {  setTitle(**"NotHelloWorld"**);  setSize(***DEFAULT\_WIDTH***, ***DEFAULT\_HEIGHT***);   MyJPanel mb = **new** MyJPanel();  add(mb);   } }  **class** MyJPanel **extends** JPanel {  MyJPanel() {  JButton myButton = **new** JButton(**"Hello"**);  myButton.addActionListener(**new** ActionListener() {  @Override  **public void** actionPerformed(ActionEvent e) {  System.***out***.println(**"Обработчик события"**);  }  });  add(myButton);  } } |

## Использовать классы – адаптеры: создать класс, расширяющий класс-адаптер, или создать сам класс - адаптер. (Здесь создается анонимный класс адаптер)

|  |
| --- |
| **import** javax.swing.\*; **import** java.awt.\*; **import** java.awt.event.ActionEvent; **import** java.awt.event.ActionListener; **import** java.awt.event.MouseAdapter; **import** java.awt.event.MouseEvent;  **public class** NotHelloWorld {  **public static void** main(String[] args) {  EventQueue.*invokeLater*(**new** Runnable() {  **public void** run() {  NotHelloWorldFrame frame = **new** NotHelloWorldFrame();  frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  frame.setVisible(**true**);  }  });  } }  **class** NotHelloWorldFrame **extends** JFrame {  **public static final int *DEFAULT\_WIDTH*** = 300;  **public static final int *DEFAULT\_HEIGHT*** = 200;   **public** NotHelloWorldFrame() {  setTitle(**"NotHelloWorld"**);  setSize(***DEFAULT\_WIDTH***, ***DEFAULT\_HEIGHT***);   NotHelloWorldComponent mb = **new** NotHelloWorldComponent();  add(mb);   } }  **class** NotHelloWorldComponent **extends** JComponent {  **public static int** *MESSAGE\_X* = 75;  **public static int** *MESSAGE\_Y* = 100;   NotHelloWorldComponent() {  addMouseListener(**new** MouseAdapter() {  @Override  **public void** mouseClicked(MouseEvent e) {  *MESSAGE\_X* = e.getX();  *MESSAGE\_Y* = e.getY();  System.***out***.println(e.getX() + **" "** + e.getY());  repaint();  }  });  }   **public void** paintComponent(Graphics g) {   g.drawString(**"Hello"**, *MESSAGE\_X*, *MESSAGE\_Y*);  }  } |

## Использовать **интерфейс Action (**действие) и реализующий его класс **AbstractAction**. Это для того, чтобы обрабатывать одну и туже команду из разных источников событий (меню, нажатие клавиш и т.д.).

|  |
| --- |
| **public class** Test {  **public static void** main(String[] args) {  EventQueue.*invokeLater*(**new** Runnable() {  **public void** run() {  SimpleWindow frame = **new** SimpleWindow();  frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  frame.setVisible(**true**);  }  });  } }  **class** SimpleWindow **extends** JFrame {  **private** ExitAction **exitAction**;  **private** DeactivateAction **deactivateAction**;   SimpleWindow() {  setSize(250, 250);  setTitle(**"Окно с меню"**);  setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);   **exitAction** = **new** ExitAction();  **deactivateAction** = **new** DeactivateAction();   JMenuBar menuBar = **new** JMenuBar();  setJMenuBar(menuBar);  JMenu fileMenu = **new** JMenu(**"Файл"**);*//панель меню* fileMenu.add(**deactivateAction**);  fileMenu.add(**exitAction**);  menuBar.add(fileMenu);   JPanel panel = **new** JPanel();*//панель кнопок* panel.add(**new** JButton(**exitAction**));  panel.add(**new** JButton(**deactivateAction**));  add(panel);  }   **class** ExitAction **extends** AbstractAction {  *//просто задаем название пункта меню и кнопки* ExitAction() {  putValue(Action.***NAME***, **"Выйти"**);  }   **public void** actionPerformed(ActionEvent event) {  System.*exit*(0);  }  }   **class** DeactivateAction **extends** AbstractAction {  *//просто задаем название пункта меню и кнопки* DeactivateAction() {  putValue(Action.***NAME***, **"Запретить выход"**);  }   **public void** actionPerformed(ActionEvent event) {  **if** (**exitAction**.isEnabled()) {  **exitAction**.setEnabled(**false**);  putValue(Action.***NAME***, **"Разрешить выход"**);   } **else** {  **exitAction**.setEnabled(**true**);  putValue(Action.***NAME***, **"Запретить выход"**);  }  }  } } |

Можно еще и так для задания действия для одной только кнопки:

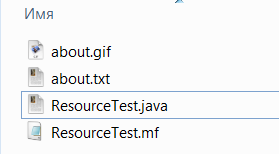
|  |
| --- |
| JMenu fileMenu = **new** JMenu(**"File"**);  fileMenu.add(**new** AbstractAction(**"Exit"**) {  **public void** actionPerformed(ActionEvent event) {  System.*exit*(0);  } }); |

Или так:

|  |
| --- |
| JCheckBoxMenuItem **readonlyItem** = **new** JCheckBoxMenuItem(**"Read-only"**);  **readonlyItem**.addActionListener(**new** ActionListener() {  **public void** actionPerformed(ActionEvent event) {  …  } }); |

# Сборка jar и размещение на сервере Tomcat (java Web Start)

## Создание jar



1. Компилируем

Javac ResourceTest.java

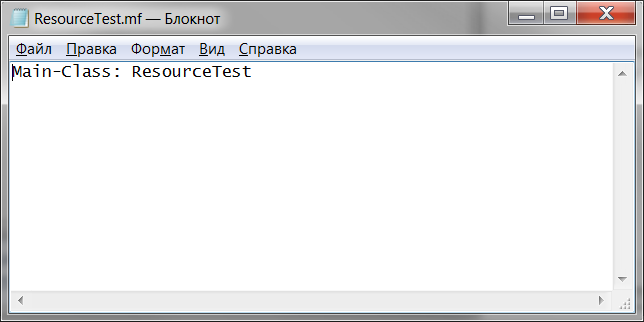
1. Создаем jar-архив:
   1. Простой:

**jar cvf ResourceTest.jar \*.class**

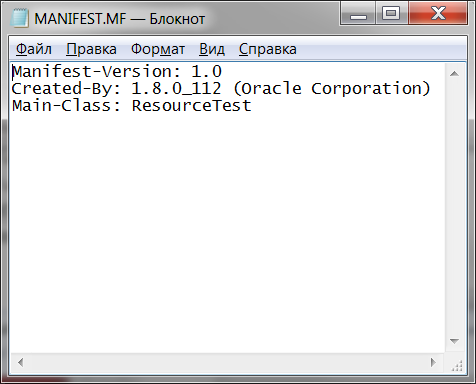
* 1. С файлами и с добавлением информации в манифест о главном классе:

**Jar cvfm ResourceTest.jar ResourceTest.mf \*.class \*.gif \*.txt**

При этом информация о главном классе должна быть в



Она добавится в jar - архив в



1. Запускаем программу из jar-архива:

Java –jar ResourceTest.jar

К ресурсам из кода обращаться можно так:

|  |
| --- |
| URL aboutURL = getClass().getResource(**"about.gif"**);  Image img = Toolkit.*getDefaultToolkit*().getImage(aboutURL);  InputStream stream = getClass().getResourceAsStream(**"about.txt"**); Scanner in = **new** Scanner(stream); |

## Размещение на сервере tomcat (java web start)

Размещение на сервере tomcat (java web start)

### Подготовить файл jnlp

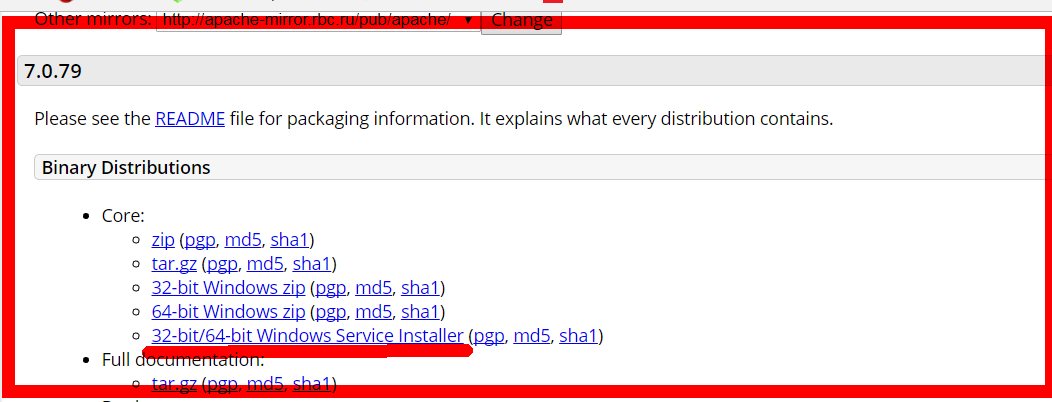
ResourceTest.jnlp:

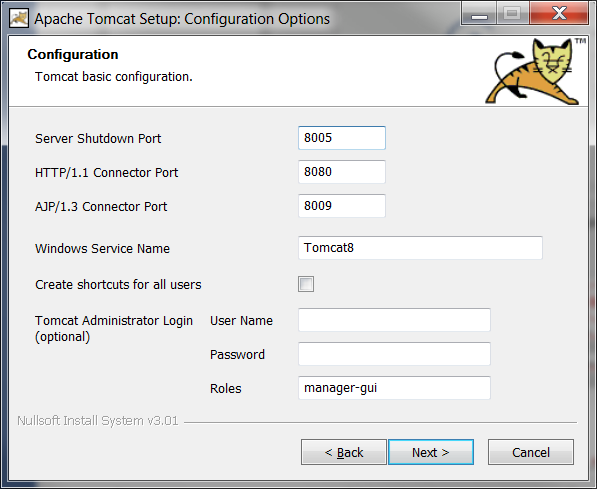
|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <jnlp spec="1.0+"codebase="http://localhost:8080/ResourceTest  href="ResourceTest.jnlp">  <information>  <title>ResourceTest Demo Application</title>  <vendor>Cay S. Horstman</vendor>  <description>A ResourceTest</description>  <offline-allowed/>  </information>  <resources>  <j2se version="5.0+"/>  <jar href="ResourceTest.jar"/>  </resources>  <application-desc/>  </jnlp> |



### Установить Tomcat:

<https://tomcat.apache.org/download-70.cgi>





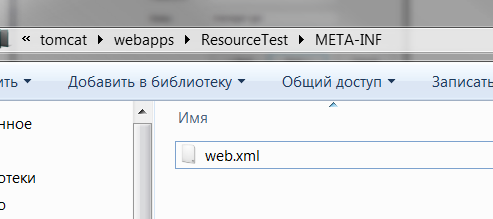
### Поместить файлы в каталог:

1. Создать каталог с названием приложения:

C:\tomcat\webapps\**ResourceTest**

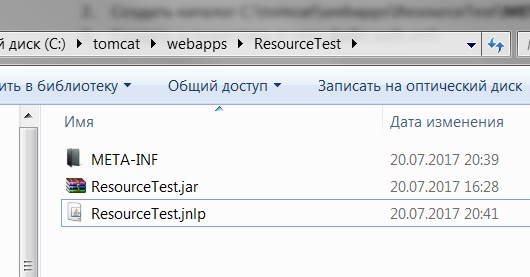
1. Создать каталог C:\tomcat\webapps\ResourceTest\**META-INF**
2. Создать и поместить в него файл web.xml:



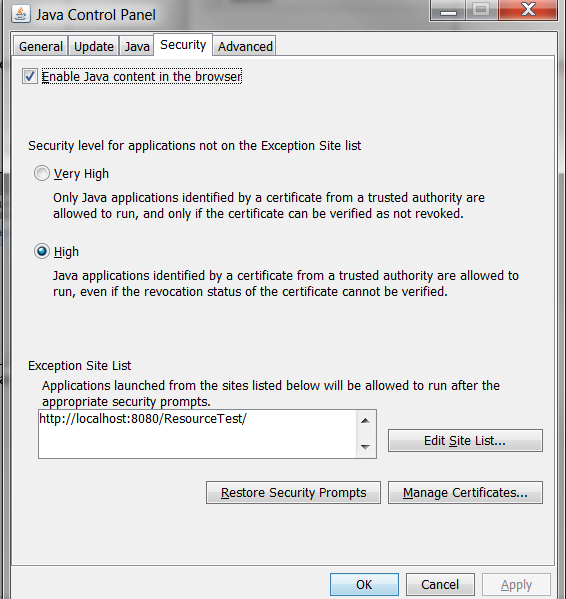


1. Поместить в каталог C:\tomcat\webapps\ResourceTest файлы: ResourceTest.jnlp и ResourceTest.jar





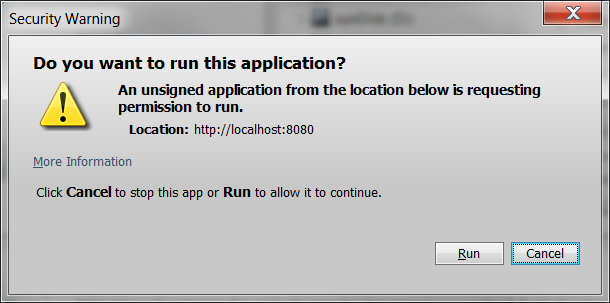
1. Настроить исключения в Java Control panel:

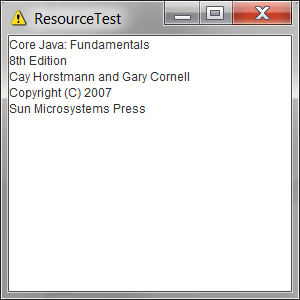


1. В браузере ввести адрес сервера и название файла jnlp:

http://localhost:8080/ResourceTest/ResourceTest.jnlp

запустить загруженный файл





**Примечание**: путь по умолчанию в браузере для сохранения файла ResourceTest.jnlp видимо содержит недопустимые символы. ☹ Так что нужно его поместить в корень диска C: и запустить.

# Applet

Пример:

|  |
| --- |
| **import** java.awt.\*; **import** javax.swing.\*;  */\*\*  \** ***@author*** *Cay Horstmann  \** ***@version*** *1.22 2007-06-12  \*/* **public class** NotHelloWorldApplet **extends** JApplet {  **public void** init() {  EventQueue.*invokeLater*(**new** Runnable() {  **public void** run() {  JLabel label = **new** JLabel(**"Not a Hello, World applet"**, SwingConstants.***CENTER***);  add(label);  }  });  } } |



Далее создаем html

|  |
| --- |
| <html>  <head>  <title>NotHelloWorldApplet</title>  </head>  <body>  <applet code="NotHelloWorldApplet.class"  width="300" height="100">  </applet>  </body>  </html> |



Компилируем и запускаем просмотрщиков апплетов или браузером:

|  |
| --- |
| appletviewer NotHelloWorldApplet.html |

# Создать свой класс исключения:

|  |
| --- |
| **class** A {  **public static void** main(String[] args) {   **try** {  **int** result = Factorial.*getFactorial*(-6);  System.***out***.println(result);  } **catch** (**FactorialException** ex) {   System.***out***.println(ex.getMessage());  System.***out***.println(ex.getNumber());  }  } }  **class** Factorial {   **public static int** getFactorial(**int** num) **throws** FactorialException {   **int** result = 1;  **if** (num < 1) **throw new** FactorialException(**"Число не может быть меньше 1"**, num);   **for** (**int** i = 1; i <= num; i++) {   result \*= i;  }  **return** result;  } }  **class FactorialException** **extends** Exception {   **private int number**;   **public** FactorialException(String message, **int** num) {   **super**(message);  **number** = num;  }   **public int** getNumber() {  **return number**;  } } |

# Диагностические утверждения

|  |
| --- |
| **public int** getFactorial(**int** num){   **assert** num>0;  … } |

Запускать командой:

Java –ea MyClass

Отключать командой:

Java –da MyClass

# Логирование

|  |
| --- |
| **class** A {  **static** Logger *log* = Logger.*getLogger*(A.**class**.getName());   **public static void** main(String[] args) {   *log*.info(**"Some message"**); |

Пример 2:

|  |
| --- |
| **class** Factorial {  **static** Logger *log* = Logger.*getLogger*(Factorial.**class**.getName());  **public static int** getFactorial(**int** num) {   **try** {  **throw new** Exception(**"Some exception"**);  } **catch** (Exception ex) {  *log*.log(Level.***SEVERE***, **"Exception: "**, ex);  } |
|  |

# StackTrace

Пример:

|  |
| --- |
| Throwable t = **new** Throwable(); StackTraceElement[] frames = t.getStackTrace(); **for** (StackTraceElement f : frames)  System.***out***.println(f); |

# Обобщенные классы (generic)

|  |
| --- |
| **public class** Pair<T>  {  **public** Pair() { **first** = **null**; **second** = **null**; }  **public** Pair(T first, T second) { **this**.**first** = first; **this**.**second** = second; }   **public** T getFirst() { **return first**; }  **public** T getSecond() { **return second**; }   **public void** setFirst(T newValue) { **first** = newValue; }  **public void** setSecond(T newValue) { **second** = newValue; }   **private** T **first**;  **private** T **second**; }  **public class** PairTest1 {  **public static void** main(String[] args) {  String[] words = {**"Mary"**, **"had"**, **"a"**, **"little"**, **"lamb"**};  System.***out***.println(ArrayAlg.*getValue*(words));  } }  **class** ArrayAlg {  **public static** String *getValue*(String[] a) {  Pair<String> p = **new** Pair<>(a[0], a[2]);  **return** p.<String>getFirst();  } } |

Типизированные методы - отдельные методы с параметрами возвращаемого типа. Он определяется в обычном классе или в обощенном

|  |
| --- |
| **class** ArrayAlg {  **public static** <T>T getMiddle(T[] a){  **return** a[0];  } }  **public class** PairTest1 {  **public static void** main(String[] args)  {  String [] str={**"232323"**,**"wwww"**, **"555555"**};  System.***out***.println(ArrayAlg.<String>*getMiddle*(str));  } } |

Задать ограничение для метода, например обобщенный метод может работать только с экземплярами классов, реализующих интерфейс Comparable:

|  |
| --- |
| **public static** <T extends Comparable>T getMin(T[] a){…} |

Подстановки с ограничениями суперклассов позволяют писать супер-класс и его наследников, а подстановки с ограничениями Наследников – читать

|  |
| --- |
| **Пример добавления наследников:**  **public class** TKO {  **public static void** main(String[] args) {  List<? **super** B> one = **new** ArrayList<B>();  *addsome*((List<? **super** B>) one);  }   **private static void** addsome(List<? **super** B> l) {  l.add(**new** B());  l.set(1, **new** C());  l.add(**new** C());l.add(**new** A());***//ошибка, т.к. A не наследник B***  } }  **class** A{  **int i** = 1; }  **class** B **extends** A {  **int i** = 2; }  **class** C **extends** B {  **int i** = 3; }  в коллекцию можно добавлять только объекты класса B либо его наследников |
| public static void getFirst (Pair<? Extends Employee> p) |

Неограниченные подстановки Pair<?>:

|  |
| --- |
| **class** PairAlg {  **public static void** swap(Pair<?> p) {  *swapHelper*(p);  }   **public static** <T> **void** swapHelper(Pair<T> p)  {  T t = p.getFirst();  p.setFirst(p.getSecond());  p.setSecond(t);  } } |

# Рефлексия

|  |
| --- |
| **class** A {  String **s** = **"hello"**;  **private int i2** = 0;  **private final int i3** = 0;   **public** String toString() {  **return** (**s** + **" "** + **i2**+ **" "** + **i3**);  } }  **public class** Test {  **public static void** main(String[] args) **throws** NoSuchFieldException, IllegalAccessException {  A a = **new** A();   Field f2 = a.getClass().getDeclaredField(**"s"**);  f2.setAccessible(**true**);  f2.set(a, **"goodbuy"**);   Field f1 = a.getClass().getDeclaredField(**"i2"**);  f1.setAccessible(**true**);  f1.set(a, 3);   Field f3 = a.getClass().getDeclaredField(**"i3"**);//ошибки не будет, но i3 не изменится  f3.setAccessible(**true**);  f3.set(a, 3);   System.***out***.println(a);  } } |
| goodbuy 3 0 |

# Потоки

|  |
| --- |
| **public class** TKO **extends** Thread {  **public static void** main(String[] args) {  MyThread t1 = **new** MyThread(1);  MyThread t2 = **new** MyThread(2);  Thread thr = **new** Thread(t1);  thr.start();  thr = **new** Thread(t2);  thr.start();   } }  **class** MyThread **implements** Runnable {  **int a**;   MyThread(**int** value) {  **a** = value;  }   @Override  **public void** run() {  **for** (**int** i = 0; i < 1000; i++) {  System.***out***.println(**a**);  **try** {  Thread.*sleep*((**int**) (Math.*random*() \* 10));  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  } } |

join – ожидает завершения потока

yield – отдает управление более приоритетным потокам

resume – Возобновляет поток (не рекомендуется)

Сделать потом-демон:

t.setDaemon(true)

## Синхронизация потоков

### Метод 1 (более новый): реентерабельная блокировка

Каждый поток должен работать со своим отдельным экземпляром класса. При этом один и тот же объект Lock можно использовать в разных методах.

Объекты условия (newCondition)

Они нужны, когда один поток получил эксклюзивный доступ, но ждет какого-либо условия, чтобы начать выполнять действия.

|  |
| --- |
| Condition **condition** = **lock**.newCondition(); |

Если поток видит, что условие не выполняется, то он вызывает **condition*.await();*** При этом данный поток деактивизирован и отдает блокировку. Сам себя он разблокировать не может. Чтобы поток снова стал активным, другой поток должен вызвать **condition*.signalAll();***

|  |
| --- |
| **import** java.util.concurrent.locks.\*;  **public class** SynchBankTest {  **public static void** main(String[] args) {  **for** (**int** i = 0; i < 5; i++) {  Thread t = **new** Thread(**new** Bank());  t.start();  }  } }  **class** Bank **implements** Runnable {   **private** Lock **lock**;  **private** Condition **condition**;   **public** Bank() {  **lock** = **new** ReentrantLock();  **condition** = **lock**.newCondition();  }   **public void** run() {  **while** (**true**) {  **try** {  transfer();  Thread.*sleep*((**int**) (100 \* Math.*random*()));  } **catch** (Exception e) {  }   }  }   **public void** transfer() **throws** InterruptedException {  **lock**.lock();  **try** {  **int** i = (**int**) Math.*random*() \* 100;  **while** (i % 3 != 0) {  **condition**.await();*//ждем условия* }  System.***out***.println(**"Hello"** + Thread.*currentThread*());  **condition**.signalAll();*//разблокировать другие потоки в этом методе* } **finally** {  **lock**.unlock();  }  }  } |

* Перед блокировкой можно проверить, может ли поток заблокировать ресурс tryLock(): - если может, то возвращает true и блокирует поток (при этом не надо отдельно вызывать lock()).

If(**bankLock.tryLock()**){…}

### Метод синхронизации 2 (простой): synchronized

|  |
| --- |
| **public synchronized void** transfer(**int** i) **throws** InterruptedException {  **while** (i % 2 != 0) {  wait();//ждем условия  }  i++;  System.***out***.print(**"Hello"**);  notifyAll();();//разблокировать другие потоки в этом методе } |

Для ожидания используется wait(). Для разблокировки ожидающих потоков notify, notifyAll

### Синхронизированные блоки

|  |
| --- |
| **class** Test {  **public static void** main(String[] args) {  One n = **new** One();  **synchronized** (n) {//синхронизированный блок  n.plus();  n.minus();  }  } }  **class** One {  **int i**=0;  **public void** plus() {  **i**++;  }  **public void** minus() {  **i**--;  } } |

### Volatile – обеспечивает атомарность поля

private volatile int I;

### Не рекоментованные методы(устаревшие):

stop, suspend, resume

## Пул потоков

Помимо интерфейса Runnable с методом run() есть интерфейс Callable с методом call(). Метод call возвращает значение:

|  |
| --- |
| **class** Test {  **public static void** main(String[] args) **throws** Exception {   One n = **new** One();  System.***out***.println(n.call());  } }  **class** One **implements** Callable<Integer> {  **private int i** = 0;   @Override  **public** Integer call() **throws** Exception {  **while** (**i** < 1000)  **i**++;  **return i**;  } } |

Интерфейс Future хранит результат асинхронного вычисления. Для него есть класс-оболочка FutureTask. FutureTask может превращать Callable одновременно в Future и Runnable:

|  |
| --- |
| **class** Test {  **public static void** main(String[] args) **throws** Exception {   One myCallable = **new** One();  FutureTask<Integer> task = **new** FutureTask<Integer>(myCallable);  Thread t = **new** Thread(task);  t.start();  System.***out***.println(task.get());  }  }   **class** One **implements** Callable<Integer>{  **private int i**=0;   @Override  **public** Integer call() **throws** Exception {  **while**(**i**<1000)  **i**++;  **return i**;  } } |

Здесь t.start(); вызывает метод call, а task.get() возвращает значение.

Пул потоков содержит множество простаивающих потоков, готовых к запуску. Мы помещаем Runnable в пул, и один из потоков вызывает его метод run(). После завершения метода run, поток не уничтожается, а остается в пуле. Класс Executors создает пулы потоков:

|  |
| --- |
| **class** Test {  **public static void** main(String[] args) **throws** Exception {   **ExecutorService service = Executors.*newCachedThreadPool*();**  **for** (**int** i = 0; i < 30; i++) {  service.**submit**(**new** One());  }  service.shutdown();  } }  **class** One **implements** Runnable {  **private static int** *j* = 1;  **private int i** = 1;   @Override  **public void** run() {  **while** (**i** < 100) {  **i**++;  *j*++;  }  System.***out***.println(*j*);  } } |

Или с использованием Callable:

|  |
| --- |
| **class** Test {  **public static void** main(String[] args) **throws** Exception {   **ExecutorService service = Executors.*newFixedThreadPool*(3)**;  **for** (**int** i = 0; i < 3; i++) {  **int** k = service.**submit**(**new** One()).**get**();  System.***out***.println(k);  }  service.shutdown();  } }  **class** One **implements** Callable<Integer> {  **private static int** *j* = 1;  **private int i** = 1;   @Override  **public** Integer call() **throws** Exception {  **while** (**i** < 100) {  **i**++;  *j*++;  }  **return** *j*;  } } |

Семафор:

|  |
| --- |
| **public class** Test {   **public static void** main(String[] args) {   Semaphore sem = **new** Semaphore(1); *// 1 разрешение* **new** Thread(**new** CountThread(sem)).start();  **new** Thread(**new** CountThread(sem)).start();  } }   **class** CountThread **implements** Runnable {  **static int** *j*=0;  **int d**=++*j*;  Semaphore **sem**;  CountThread(Semaphore sem) {  **this**.**sem**=sem;  }   **public void** run() {   **try** {  System.***out***.println(**"Thread "**+**d** +**" ожидает разрешение"**);  **sem**.acquire();  **for** (**int** i = 1; i < 5; i++) {  System.***out***.println(**"Thread "**+**d**);  Thread.*sleep*(100);  }  } **catch** (InterruptedException e) {  System.***out***.println(e.getMessage());  }  System.***out***.println(**"Thread "**+**d** + **" освобождает разрешение"**);  **sem**.release();  } } |

# **Потоки и файлы**

Абстрактные файлы **InputStream** и **OutputStream** – основа иерархии классов ввода/вывода. Выполняют считывание или запись отдельных байтов или массива байтов.

Абстрактные классы **Reader** и **Write** – отдельная иерархия для работы с Unicode-символами.

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | **InputStream** | Метод | **OutputStream** |
| abstract int read() | Возвращает либо считанный байт, либо -1 | abstract void write(int b) | Записывает 1 байт в выходные данные |
| int read(byte[]b) | Считывает данные длиной (b.length) в байтовых массив | void write(byte[]b) | Записывает все байты из массива b |
| int read(byte[]b, int off, int len) | off – позиция смещения в массиве b, с которой в него будут считаны данные. len – макс кол-во считываемых байтов | void write(byte[]b, int off, int len) | Записывает диапазон байтов из массива b |
| long skip (long n) | Сколько кол-во байтов нужно пропустить перед началом чтения | void flush() | Направляет данные из буфера в их место назначение, очищая буфер |

После использования потоков обязательно нужно вызывать close(). Иерархию классов см. том 2 стр 23

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | **Reader** | Метод | **Writer** |
| abstract int read() | Возвращает либо считанный байт, либо -1 | abstract void write(int с) | Записывает 1 байт в выходные данные |

Иерархию классов см. том 2 стр 24

Пример чтения из файла и запись в другой файл (FileInputStream, FileOutputStream):

|  |
| --- |
| **public class** Test{  **public static void** main(String[] args) **throws** IOException {  *//создаем входной поток* **FileInputStream** fin = **new FileInputStream**(**"N:\\подготовка\\java\\horstmann\\"** +  **"CoreJavaBook\\v1ch14\\AlgorithmAnimation\\AlgorithmAnimation.java"**);   **byte**[] b = **new byte**[10000];*//массив для считывания в него* **int** i = 0;  **while** ((b[i] = (**byte**) fin.read()) != -1) {  System.***out***.print((**char**)b[i]);  i++;  }  fin.close();  *//создаем выходной поток* FileOutputStream fout = **new** FileOutputStream(**new** File(**"N:\\подготовка\\java\\horstmann\\"** +  **"CoreJavaBook\\v1ch14\\AlgorithmAnimation\\testFile.txt"**));  fout.write(b);*//записываем из выходного потока в файл* fout.close();  } } |

Нужно использовать фильтры(последовательности вызовов потоков) для более удобного представления считываемых данных:

|  |
| --- |
| DataInputStream din = **new** DataInputStream(  **new** BufferedInputStream(**new** FileInputStream(**"file.txt"**))  ); |

Здесь:

DataInputStream - для считывания всех простых типов данных java) для того, чтобы из входного потока достать только числа.

BufferedInputStream – помещает данные в буфер, а не каждый раз запрашивает их у операционной системы как FileInputStream.

## Текстовые данные.

### Запись текстовых данных.

Для них требуется указание кодировки, или используется кодировка по-умолчанию.

|  |
| --- |
| **import** java.io.\*;  **public class** Test {  **public static void** main(String[] args) **throws** IOException {  *//создаем входной поток* FileInputStream fin = **new** FileInputStream(**"N:\\подготовка\\java\\horstmann\\"** +  **"CoreJavaBook\\v1ch14\\AlgorithmAnimation\\AlgorithmAnimation.java"**);  **InputStreamReader** in = **new InputStreamReader**(fin, **"ISO8859\_5"**);   **int** ch;  **char**[] chars = **new char**[10000];  **int** i = 0;   **while** ((ch = in.read()) != -1) {  chars[i] = (**char**) in.read();  System.***out***.print(chars[i++]);  }   *//запись в файл1:* **OutputStreamWriter** or = **new OutputStreamWriter**(**new** FileOutputStream(**new** File(**"N:\\подготовка\\java\\"** +  **"horstmann\\CoreJavaBook"** +  **"\\v1ch14\\AlgorithmAnimation\\file1.txt"**)));  or.write(chars);   *//запись в файл2:* **FileWriter** fr = **new FileWriter**(**new** File(**"N:\\подготовка\\java\\horstmann\\CoreJavaBook"** +  **"\\v1ch14\\AlgorithmAnimation\\file2.txt"**));  fr.write(chars);  *//запись в файл3:* **PrintWriter** pr = **new PrintWriter**(**new** File(**"N:\\подготовка\\java\\horstmann\\CoreJavaBook"** +  **"\\v1ch14\\AlgorithmAnimation\\file3.txt"**));  pr.write(chars);  //добавление в файл текста  pr.print(**"hello"**);  pr.println(**"\n good buy"**);  fin.close();  in.close();  or.close();  fr.close();  pr.close();  } } |

**PrintWriter –** лучше всего использовать для записи

### **Считывание текстовых данных.**

1-ый вариант см в предыдущем примере.

2-ой вариант:

|  |
| --- |
| **public class** Test {  **public static void** main(String[] args) **throws** IOException {  **BufferedReader** in = **new BufferedReader**(**new** FileReader(**"N:\\подготовка\\java\\horstmann\\CoreJavaBook\\v1ch14\\AlgorithmAnimation\\AlgorithmAnimation.java"**));   String line=**""**;  **while**((line=in.readLine())!=**null**){  System.***out***.println(line);  }  } } |

3-ий вариант(предпочтителдьный)

|  |
| --- |
| //чтение из консольного ввода  Scanner in = **new** Scanner(System.***in***); System.***out***.println(**"Как вас зовут"**); System.***out***.println(in.next()) |
| //чтение из файла  Scanner in = **new** Scanner(**new** File(**"N:\\new 1.txt"**)); **while** (in.hasNext()) {  System.***out***.println(in.next());  } |

Сканировать строку и распарсить:

String line = in.nextLine();  
String[] tokens = line.split(**"\\|"**);

## **Двоичные данные.**

Интерфейсы DataOutput и DataInput позволяют записывать двоичные данные в файл и считывать двоичные данные из файла:

|  |
| --- |
| **public class** Test {  **public static void** main(String[] args) **throws** IOException {   **int** i=10001;  Short s=1;  Long l = 100L;  **float** f = 12.32f;  **double** d = 123.2112;  **char** c=**'a'**;  String str=**"привет!!!"**;  //поток для записи данных в двоичный файл  DataOutput dout = **new** DataOutputStream(**new** FileOutputStream(**new** File(**"C:\\test.bin"**)));  dout.writeInt(i);  dout.writeShort(s);  dout.writeLong(l);  dout.writeFloat(f);  dout.writeDouble(d);  dout.writeChar(c);  dout.writeUTF(str); //поток для чтения данных из двоичного файла  DataInput din = **new** DataInputStream(**new** FileInputStream(**"C:\\test.bin"**));  System.***out***.println(din.readInt());  System.***out***.println(din.readShort());  System.***out***.println(din.readLong());  System.***out***.println(din.readFloat());  System.***out***.println(din.readDouble());  System.***out***.println(din.readChar());  System.***out***.println(din.readUTF());   } } |

При этом последовательность считывания должна соответствовать последовательности записи.

### **Произвольный доступ к файлам –** можно как читать, так и записывать(это указывается во втором параметре)

RandomAccessFile in = **new** RandomAccessFile(**"test.txt"**,**"r"**);  
RandomAccessFile inOut = **new** RandomAccessFile(**"test.txt"**,**"rw"**);

У этих объектов есть file pointer (указатель). Метод seek() устанавливает указатель на произвольную байтовую позицию внутри файла. getFilePointer() – возвращает текущую позицию. RandomAccessFile реализует интерфейсы DataInput и DataOutput.

------------------------------чтение из файла, кот находится там же, где и класс программы

InputStream stream = getClass().getResourceAsStream(**"about.txt"**);  
Scanner in = **new** Scanner(stream);  
**while** (in.hasNext())  
 textArea.append(in.nextLine() + **"\n"**);

**Ввод**

InputStream in = contents.getInputStream();  
BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(in));  
String line;  
**while** ((line = reader.readLine()) != **null**)  
{  
 **panel**.append(line);  
 **panel**.append(**"\n"**);  
}

**Вывод**

ByteArrayOutputStream out = **new** ByteArrayOutputStream();  
PrintStream printOut = **new** PrintStream(out);  
printOut.print(**panel**.getText());

Запись в файл:

FileOutputStream fout = **new** FileOutputStream(**"MyProperties.properties"**);  
fout.write((**"привет!!"**.getBytes()));

Чтение файла:

FileInputStream fin = **new** FileInputStream(**"MyProperties.properties"**);  
  
**int** i=0;  
**while**((i=fin.read())!=-1){  
 System.***out***.print((**char**)i);  
}

### **zip – архивы**

**ZipInputStream/ZipOutputStream**

|  |
| --- |
| **public class** Test {   **public static void** main(String[] args) **throws** Exception { *//создание zip-архива* ZipOutputStream zos = **new** ZipOutputStream(**new** FileOutputStream(**"ZipTest.zip"**));   **for** (**int** i = 0; i++ < 3; ) {  *// создание файла;* zos.putNextEntry(**new** ZipEntry(**"hello-world\_"** + i + **".txt"**));  *//запмсь данных в файл* zos.write((**"данные в файле"** + i + **"\n"**).getBytes());  zos.write(**"Hello World!"**.getBytes());  *// not available on BufferedOutputStream* zos.closeEntry();  }  zos.close(); *//чтение zip-архива  //поток для чтения zip-архива* ZipInputStream zin = **new** ZipInputStream(**new** FileInputStream(**"ZipTest.zip"**));  ZipEntry entry;*//получает имя файла в архиве* **while** ((entry = zin.getNextEntry()) != **null**) {  System.***out***.println(**"File name: "** + entry.getName());  Scanner in = **new** Scanner(zin);*//поток для чтения содержимого файла* **while** (in.hasNextLine()) {  System.***out***.println(in.nextLine());  }  zin.closeEntry();  System.***out***.println(**"==========="**);  }  } } |

## **Сериализация.**

ObjectOutputStream/ ObjectInputStream

|  |
| --- |
| **public class** Test {   **public static void** main(String[] args) **throws** IOException, ClassNotFoundException {  MySerialize ms1 = **new** MySerialize(1,**"object 1"**);  MySerialize ms2 = **new** MySerialize(2,**"object 2"**);  MySerialize ms3 = **new** MySerialize(3,**"object 3"**);    **ObjectOutputStream** out = **new** ObjectOutputStream(**new** FileOutputStream(**"object.obj"**));  out.**writeObject** (ms1);  out.**writeObject** (ms2);  out.**writeObject** (ms3);  out.close();   **ObjectInputStream** in = **new** ObjectInputStream(**new** FileInputStream(**"object.obj"**));  MySerialize ms4=(MySerialize)in.**readObject**();  MySerialize ms5=(MySerialize)in.**readObject**();  MySerialize ms6=(MySerialize)in.**readObject**();  in.close();  List<MySerialize> list = **new** LinkedList<MySerialize>();  list.add(ms1); list.add(ms2); list.add(ms3); list.add(ms4);  list.add(ms5); list.add(ms6);  **for**(MySerialize l: list){  System.***out***.println(l.**i**+**" "**+l.**str**);  }   } }  **class** MySerialize **implements Serializable**{  **int i**=0;  String **str**=**""**;  MySerialize(**int** i, String str){  **this**.**i**=i;  **this**.**str**=str;  } } |

**transient** – исключает поля класса из сериализации.

## **Отображаемые в памяти файлы.**

|  |
| --- |
| **public class** Test {   **public static void** main(String[] args) **throws** IOException {  *//для считывания из файла в канал* RandomAccessFile rf = **new** RandomAccessFile(**"c:\\file.txt"**, **"rw"**);  FileChannel fromChannel = rf.getChannel();   String str = **" good buy"**;  *//записываем в канал строчку good buy в 5-ую позицию* fromChannel.write(ByteBuffer.*wrap*(str.getBytes()),5L);   *//Создаем второй канал для записи в другой файл* FileOutputStream toFileStream = **new** FileOutputStream(**new** File(**"c:\\file2.txt"**));  FileChannel toChannel = toFileStream.getChannel();   **try** {  *//записываем из одного канала в другой* fromChannel.transferTo(0, fromChannel.size(), toChannel);  } **finally** {  **if** (fromChannel != **null**)  fromChannel.close();  **if** (toChannel != **null**)  toChannel.close();  rf.close();  toFileStream.close();  }   } } |

RandomAccessFile – позволяет как считывать, так и записывать данные в канал, а FileOutputStream только записывать. В итоге в обоих файлах **file.txt и file2.txt** будут одинаковые измененные данные.

## **Регуляные выражения.**

|  |
| --- |
| **public class** Test {   **public static void** main(String[] args) **throws** IOException {  Pattern pt = Pattern.*compile*(**"[a-z]{1,}7"**);  Matcher m = pt.matcher(**"asddfg7"**);  System.***out***.println(m.matches());   *//Pattern.CASE\_INSENSITIVE - отыскивает совпадения в тексте* Pattern pt2 = Pattern.*compile*(**"[a-z]{1,}7"**, Pattern.***CASE\_INSENSITIVE***);  String str = **"123dfg7vbb435345f7"**;  Matcher m2 = pt.matcher(str);  **while** (m2.find()) {  **int** start = m2.start();  **int** end = m2.end();  System.***out***.println(str.substring(start, end));  }  } } |
| **true**  **dfg7**  **f7** |

# **XML**

Есть древовидные анализаторы, например DOM и потоковые, например SAX. Рассмотрим древовидный

|  |
| --- |
| **public class** Test {   **public static void** main(String[] args) **throws** ParserConfigurationException, IOException, SAXException {  DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();  DocumentBuilder builder = factory.newDocumentBuilder();  InputStream in = **new** FileInputStream(**new** File(**"N:\\подготовка\\java\\horstmann\\CoreJavaBook\\v2ch02\\1\\test.xml"**));  Document doc = builder.parse(in);  *//получить главный эелемент* Element root = doc.getDocumentElement();  System.***out***.println(root.getTagName());  *//получить список дочерних элементов главного элемента* NodeList list = root.getChildNodes();  *//вызываем рекурсивную функцию для обхода всех элементов и их параметров* Test t = **new** Test();  t.recur(list);   in.close();  }   **public void** recur(NodeList list) {  *// от 0 до конца списка элементов данного уровня* **for** (**int** i = 0; i < list.getLength(); i++) {  Node child = list.item(i);*//получаем текущий узел* **if** (child **instanceof** Element) {*//если это элемент, а не текст или какой-нибудь отступ, то...* Element childElement = (Element) child;*//получаем текущий элемент* System.***out***.println(**" "** + childElement.getTagName());*//его название* NamedNodeMap attributes = childElement.getAttributes();*//список атрибутов* **for** (**int** j = 0; j < attributes.getLength(); j++) {*//идем по списку аттрибутов* Node attributeNode = attributes.item(j);  *//печатает название аттрибута и его значение* System.***out***.println(**" "** + attributeNode.getNodeName() + **": "** + attributeNode.getNodeValue());  }  *//проверяем, есть ли у текущего узла дочерние узды.  //если есть, то опять вызываем данную функцию со списков дочерних узлов.* NodeList childlist = child.getChildNodes();  **if** (childlist.getLength() > 0) recur(childlist);   }  }  } } |

Тестовый файл с xml: 

# **Подпись данных**

Основные классы:

* **MessageDigest** – для формирования отпечатка
* **KeyStore** – для загрузки хранилища ключей
* **PrivateKey** – для загрузки приватного ключа
* **Signature** – для подписи данных приватным ключом

|  |
| --- |
| **public class** DocSign {  **public static void** main(String[] args){  *//считываем в массив объект для подписи:* Path path = Paths.*get*(**"N:\\подготовка\\java\\horstmann\\MyHelp3.docx"**);  **byte**[] data = Files.*readAllBytes*(path);   *//пример 1: формируем отдельно отпечаток, а зетем отдельно подписываем его  //выбираем алгоритм отпечатка и формируем отпечаток* MessageDigest md = MessageDigest.*getInstance*(**"SHA-512"**);  md.update(data);  *//загружаем из хранилище ключей закрытый ключ* FileInputStream fin = **new** FileInputStream(**"N:\\подготовка\\java\\horstmann\\138860177375.p12"**);  KeyStore keyFile = KeyStore.*getInstance*(**"PKCS12"**);  keyFile.load(fin, **"qwerty"**.toCharArray());*//доступ к хранилищу* PrivateKey key = (PrivateKey) keyFile.getKey(**"138860177375"**, **"qwerty"**.toCharArray());*//получение закр ключа  //подписываем:* Signature sign = Signature.*getInstance*(**"NONEwithRSA"**);*//алгоритм подписи* sign.initSign(key);*//инициализация закрытого ключа* sign.update(md.digest());*//подпись отпечатка* **byte**[] signature = sign.sign();*//запись в байтовый массив подписи* signature = Base64.*getEncoder*().encode(signature);*//преобразуем подпись в BASE64* String s = **new** String(signature, **"UTF8"**);  System.***out***.println(s);   *//пример 2: Не делаем отдельно отпечаток, а сразу подписываем* Signature sign2 = Signature.*getInstance*(**"SHA512withRSA"**);*//алгоритм подписи* sign2.initSign(key);*//инициализация закрытого ключа* sign2.update(data);*//подпись данных(отпечатка нет)* **byte**[] signature2 = sign2.sign();*//запись в байтовый массив подписи* signature2 = Base64.*getEncoder*().encode(signature2);*//преобразуем подпись в BASE64* String s2 = **new** String(signature2, **"UTF8"**);  System.***out***.println(s2);  } } |



# **Java 8**

## Анонимные методы (лямбда-выражения) и дефолтные методы

|  |  |
| --- | --- |
| С лямбда | Обычная реализация |
| **interface** MyInterface {  **public void** myInt(**int** x, **int** y);   *//default - метод* **default double** sqrt(**double** x) {  **return** Math.*sqrt*(x);  } }  **public class** Test {   **public static void** main(String[] args) **throws** Exception {  *//пример Лямбда 1* **Runnable r = () ->** System.***out***.println(**"Hello Runnable"**);  Thread t = **new** Thread(r);  t.start();  *//пример Лямбда 2* **MyInterface n = (a, b) ->** {  System.***out***.println(**"a+b="** + (a + b));  System.***out***.println(**"a\*b="** + (a \* b));  System.***out***.println(**"a/b="** + (a / b));  System.***out***.println(**"-----------"**);  };  n.myInt(3, 5);  n.myInt(1000, 2000);   *//пример Лямбда 3* List<String> names = Arrays.*asList*(**"peter"**, **"anna"**, **"mike"**, **"xenia"**);  Collections.*sort*(names, **(String a, String b) ->** {  **return** a.compareTo(b);   });  *//или еще короче // Collections.sort(names, (a,b) -> a.compareTo(b));*  System.***out***.print(names);   *//вызов дефолтного метода* System.***out***.println(n.sqrt(9));  } } | **Runnable r = new Runnable()** {  @Override  **public void run()** {  System.***out***.println(**"Hello Runnable"**);  } }; Thread t = **new** Thread(r); t.start();  **MyInterface n = new MyInterface()** {  @Override  **public void myInt(int a, int b)** {  System.***out***.println(**"a+b="** + (a + b));  System.***out***.println(**"a\*b="** + (a \* b));  System.***out***.println(**"a/b="** + (a / b));  System.***out***.println(**"-----------"**);  } };  Collections.*sort*(names, **new Comparator<String>()** {  @Override  **public int compare(String a, String b)** {  **return** a.compareTo(b);  } }); |

|  |
| --- |
| **public class** Test **extends** JFrame {   Test() {  setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);  setLayout(**new** FlowLayout());  setSize(300, 200);  setVisible(**true**); *//без лямбда - выражения* JButton b1 = **new** JButton(**"Without lambda"**);  add(b1);  b1.addActionListener(**new** ActionListener() {  @Override  **public void** actionPerformed(ActionEvent e) {  System.***out***.println(**"The button 1 has been clicked!"**);  }  });   *//с лямбда - выражением* JButton b2 = **new** JButton(**"Lambda"**);  add(b2);   b2.addActionListener((ActionEvent e) ->  System.***out***.println(**"The button 2 has been clicked!"**));   }   **public static void** main(String[] args) **throws** IOException {  **new** Test();   } } |

## Функциональные интерфейсы и ссылки на методы:

Пример 1:

|  |
| --- |
| **interface** Hello{  String get(String str); }  **class** MyDate{  **static** String getDate(String name){  **return "Hello "**+name;  } }  **public class** Test {   **public static void** main(String[] args) **throws** Exception {  *//реализация функционального интерфейса через лямбда*  *//метод getDate - статический*  Hello hello2 = (name)->MyDate.*getDate*(name);  System.***out***.println(hello2.get(**"Vacia"**));  *//с использованием статических ссылок наметоды*  Hello hello = MyDate::*getDate*;  System.***out***.println(hello.get(**"Juriy"**));  } } |

Привер 2 (тоже самое, но ссылки на обчные методы, а не статические)

|  |
| --- |
| **interface** Hello{  String get(String str); }  **class** MyDate{  String getDate(String name){  **return "Hello "**+name;  } }  **public class** Test {   **public static void** main(String[] args) **throws** Exception {  MyDate myDate = **new** MyDate();   Hello hello = myDate::getDate;  System.***out***.println(hello.get(**"Juriy"**));   Hello hello2 = (name)->myDate.getDate(name);  System.***out***.println(hello2.get(**"Vacia"**));   } } |

Привер 3 (тоже самое)

|  |
| --- |
| @FunctionalInterface **interface** Converter<F, T> {  T convert(F from); }  **public class** Test {   **public static void** main(String[] args) **throws** Exception {  *//реализация функционального интерфейса через лямбда*  *//метод valueOf - статический* Converter<String, Integer> converter = (from) -> Integer.*valueOf*(from)+1;  Integer converted = converter.convert(**"123"**);  System.***out***.println(converted); *// 124   //с использованием статических ссылок на методы* Converter<String, Integer> converter2 = Integer**::***valueOf*;  Integer converted2 = converter2.convert(**"123"**);  System.***out***.println(converted2); *// 123* } } |

## Ссылки на конструктор

|  |
| --- |
| **class** Person {  String **firstName**, **lastName**;   Person(String firstName, String lastName) {  **this**.**firstName** = firstName;  **this**.**lastName** = lastName;  } }  **interface** PersonFactory<P **extends** Person> {  P create(String firstName, String lastName); }  **public class** Test {   **public static void** main(String[] args) **throws** Exception {  PersonFactory<Person> p = Person::**new**;  Person person = p.create(**"one"**, **"two"**);  System.***out***.println(person.**firstName** + **" "** + person.**lastName**);*//one two* } } |

## Предикаты— это функции, принимающие один аргумент, и возвращающие значение типа boolean. Интерфейс содержит различные методы по умолчанию, позволяющие строить сложные условия (and, or, negate).

|  |
| --- |
| **public class** Test {   **public static void** main(String[] args) **throws** Exception {  Predicate<String> predicate = (s) -> s.length() > 0;  System.***out***.println(predicate.test(**"foo"**));*// true* System.***out***.println(predicate.negate().test(**"foo"**)); *// false* Predicate<String> isEmpty = String::isEmpty;  System.***out***.println(**""**.isEmpty());*// true* } } |

## Потоки

Тип java.util.Stream представляет собой последовательность элементов, над которой можно производить различные операции. Операции над потоками бывают или промежуточными (intermediate) или конечными (terminal). Конечные операции возвращают результат определенного типа, а промежуточные операции возвращают тот же поток.  Поток создаются на основе источников, например типов, реализующих java.util.Collection, такие как списки или множества

|  |
| --- |
| **class** MyPrint{  **static void** print(String str){  System.***out***.println(str);  } }  **public class** Test {   **public static void** main(String[] args) **throws** Exception {  *//Сперва создадим источник в виде списка строк:* List<String> stringCollection = **new** ArrayList<>();  stringCollection.add(**"ddd2"**);  stringCollection.add(**"aaa2"**);  stringCollection.add(**"bbb1"**);  stringCollection.add(**"aaa1"**);  stringCollection.add(**"bbb3"**);  stringCollection.add(**"ccc"**);  stringCollection.add(**"bbb2"**);  stringCollection.add(**"ddd1"**);   stringCollection  *//создадим поток* .stream()  *//Операция Filter принимает предикат, который фильтрует все элементы потока. Эта операция является промежуточной, т.е. позволяет нам вызвать другую операцию (например, forEach) над результатом.* .filter((s) -> s.startsWith(**"a"**))  *//Операция Sorted является промежуточной операцией, которая возвращает отсортированное представление потока.* .sorted()  *//Операция map преобразовывает каждый элемент в другой объект при помощи переданной функции* .map(String::toUpperCase)  *//ForEach принимает функцию, которая вызывается для каждого элемента в (уже отфильтрованном) поток. ForEach является конечной операцией. Она не возращает никакого значения, поэтому дальнейший вызов потоковых операций невозможен.* .forEach(MyPrint::*print*);  } } |

## Параллельные потоки

|  |
| --- |
| **class** MyPrint {  **static void** print(String str) {  System.***out***.println(str);  } }  **public class** Test {   **public static void** main(String[] args) **throws** Exception {  *// Сперва создадим большой список из уникальных элементов:* **int** max = 1000000;  List<String> values = **new** ArrayList<>(max);  **for** (**int** i = 0; i < max; i++) {  UUID uuid = UUID.*randomUUID*();  values.add(uuid.toString());  }  *//Теперь измерим время сортировки этого списка.* **long** t0 = System.*nanoTime*();  *// values.stream().sorted().forEach(MyPrint::print);* values.**parallelStream**().sorted().forEach(MyPrint::*print*);   **long** t1 = System.*nanoTime*();   **long** millis = TimeUnit.***NANOSECONDS***.toMillis(t1 - t0);  System.***out***.println(String.*format*(**"parallel sort took: %d ms"**, millis));  } } |

## Ассоциативные массивы

|  |
| --- |
| **class** MyPrint {  **static void** print(String str) {  System.***out***.println(str);  }   **static void** print2(Integer id, String val) {  System.***out***.println(id + **" "** + val);  } }  **public class** Test {   **public static void** main(String[] args) **throws** Exception {  Map<Integer, String> map = **new** HashMap<>();   **for** (**int** i = 0; i < 10; i++) {  map.put(i, **"val"** + i);  }   map.forEach((id, val) -> System.***out***.println(id + **" "** + val)); *// map.forEach((id, val) -> MyPrint.print(id+" "+val)); // map.forEach((MyPrint::print2));* } } |

# **Локализация**

|  |
| --- |
| **class** Test {  **public static void** main(String args[]) **throws** IOException, SQLException {    Locale[] log = DateFormat.*getAvailableLocales*();//получить региональные настройки  **for**(Locale l: log){  Locale.*setDefault*(l);  NumberFormat f = NumberFormat.*getCurrencyInstance*(l);  **double** d=12345.5412;  String TextNumber=f.format(d).trim();  //вывести название региона и число в ее формате  System.***out***.println(l.getDisplayName()+**": "**+TextNumber);  //обратный процесс: распарсить строку, содержащее число в определенной локали  Number input=f.parse(TextNumber);  System.***out***.println(input);  }  } } |
| **Bahasa Indonesia (Indonesia): Rp12.345,54**  **12345.54**  **English (United Kingdom): £12,345.54**  **12345.54**  **…** |