Лекция 5
Программирование на Java

ФИО преподавателя: Зорина Наталья Валентиновна

e-mail: zorina@mirea.ru,zorina_n@mail.ru

Тема лекции:

Тема: Продолжение GUI
Исключения и их обработка в Java

Содержание

- Обработка событий
- Исключения
- Выражения try – catch
- Классы исключений
- I/O исключения

Продолжение GUI

Обработка событий

События и слушатели

- Стандартная библиотека классов Java содержит несколько классов, которые представляют собой типичные события
- Компоненты, такие как кнопки, генерируют событие, когда оно происходит
- Объект слушателя "ждет” события, которое должно произойти и реагирует соответствующим образом
- Мы можем конструировать объекты слушателя и принимать любые действия, которые уместны при возникновении события

Обработка событий

- Полиморфизм играет важную роль в развитии графического пользовательского интерфейса Java
- Как мы уже видели, мы устанавливаем связь между компонентом и слушателя:

```
JButton button = new JButton();
button.addActionListener(new MyListener());
```

- Заметьте, что метод `addActionListener` принимает объект `MyListenerв` качестве параметра
- На самом деле, мы можем передать методу `addActionListener` любой объект, который реализует интерфейс `ActionListener`

Обработка событий

- Код метода `addActionListener` принимает параметр `ActionListener` (интерфейс)
- Из-за полиморфизма, любой объект, который реализует этот интерфейс совместим с параметром ссылочной переменной
- Компонент может вызывать метод `actionPerformed` из-за связи между классом слушателем и интерфейсом
- Расширение класса адаптера для создания слушателя представляет собой такую же ситуацию; класс адаптера уже реализует соответствующий интерфейс

Кнопки

- Нажатая кнопка является компонентом, который позволяет пользователю инициировать действия, а именно нажав графическую кнопку с помощью мыши
- Нажатие кнопки объекта класса `Jbutton`
- Он генерирует события действия
- Пример `PushCounter` отображает кнопки и увеличивает счетчик каждый раз, когда кнопка нажимается

PushCounterPanel.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PushCounterPanel extends JPanel  {
   private int count;
   private JButton push;
   private JLabel label;
   public PushCounterPanel () {
      count = 0;
      push = new JButton ("Push Me!");
      push.addActionListener (new ButtonListener());
      label = new JLabel ("Pushes: " + count);
      add (push);
      add (label);
}
```

PushCounterPanel.java

```
 setPreferredSize (new Dimension(300, 40));
 setBackground (Color.cyan);
private class ButtonListener implements ActionListener    {
      public void actionPerformed (ActionEvent event)
    {
          count++;
          label.setText("Pushes: " + count);
       }
   }
}
```

PushCounter.java

```
import javax.swing.JFrame;
public class PushCounter
{
    public static void main (String[ ] args)
    {
        JFrame frame = new JFrame ("Push Counter");
        frame.setDefaultCloseOperation
  (JFrame.EXIT_ON_CLOSE);


        frame.getContentPane().add(new
  PushCounterPanel());



  frame.pack();
        frame.setVisible(true);
    }
}
```

Пример Push Counter

- Компонентами графического интерфейса являются кнопки, ярлыки(подписи), чтобы отобразить счетчик, панель для организации компонентов, и главное окно
- Класс `PushCounterPanel` класс представляет собой панель, которая используется для отображения кнопки и подписи
- А `PushCounterPanel` наследуется от `Jpanel`
- Конструктор `PushCounterPanel` устанавливает элементы графического интерфейса пользователя и инициализирует счетчик на ноль

Пример Push Counter

- Класс `ButtonListener` является слушателем для события действия, которые генерируется с помощью кнопки
- Он реализован как внутренний класс, это означает, что этот класс определен внутри тела другого класса
- Это облегчает коммуникацию между слушателем и компонентами GUI
- Внутренние классы должны использоваться только в тех ситуациях, когда существует тесная связь между этими двумя классами, и внутренний класс не требуется в любом другом контексте

Реализация PushCounter

- класс Слушатель написан посредством реализации интерфейса слушателя
- Класс `ButtonListener` реализует интерфейс `ActionListener`
- Интерфейс представляет собой список методов, которые должны определяться в классе реализации
- Единственный метод интерфейса `ActionListener` является метод `actionPerformed`
- Библиотека Java классов содержит интерфейсы для многих типов событий

Реализация PushCounter

- Конструктор `PushCounterPanel`:
  - instantiates the `ButtonListener` object
  - устанавливает связь между кнопкой и вызовом слушателя `addActionListener`
- Когда пользователь нажимает кнопку, компонент кнопки создает объект `ActionEvent` и вызывает методы слушателя `actionPerformed`
- Метод `actionPerformed` увеличивает счетчик и сбрасывает текст

Текстовые поля

- Давайте посмотрим на другой пример GUI, который использует другой тип компонента
- Текстовое поле позволяет пользователю ввести одну строку ввода
- Если курсор находится в текстовом поле, компонент текстовое поле генерирует событие действий, когда клавиша ввода нажата

FahrenheitPanel.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class FahrenheitPanel extends JPanel {
   private JLabel inputLabel, outputLabel, resultLabel;
   private JTextField fahrenheit;

   public FahrenheitPanel()  {
      inputLabel = new JLabel ("Enter Fahrenheit temperature:");
      outputLabel = new JLabel ("Temperature in Celsius: ");
      resultLabel = new JLabel ("---");
```

FahrenheitPanel.java

```
fahrenheit = new JTextField (5);
fahrenheit.addActionListener (new TempListener());

add (inputLabel);
add (fahrenheit);
add (outputLabel);
add (resultLabel);

setPreferredSize (new Dimension(300, 75));
setBackground (Color.yellow);
   } //end of constructor FahrenheitPanel
```

FahrenheitPanel.java

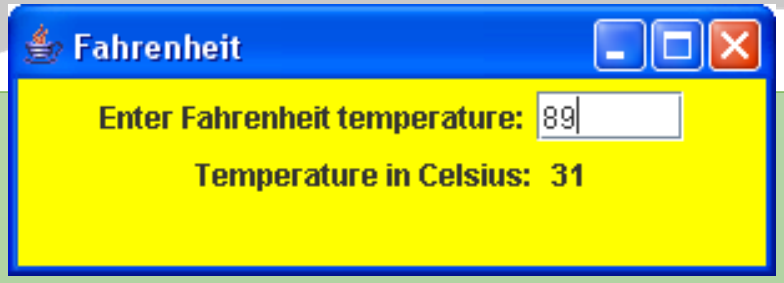
```
 // Представляет слушателя действия для поля ввода температуры.
private class TempListener implements ActionListener    {
   // Выполняет преобразование при нажатии клавиши ввод (enter) в
  // текстовое поле
public void actionPerformed (ActionEvent event)       {
         int fahrenheitTemp, celsiusTemp;

         String text = fahrenheit.getText();

         fahrenheitTemp = Integer.parseInt (text);
         celsiusTemp = (fahrenheitTemp-32) * 5/9;

         resultLabel.setText (Integer.toString (celsiusTemp));
      }
   }
}
```

Fahrenheit.java




```
import javax.swing.JFrame;

public class Fahrenheit
{
 //-----------------------------------------------------------------
// Создает и отображает графический интерфейс для преобразования
//температуры.
//-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Fahrenheit");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      FahrenheitPanel panel = new FahrenheitPanel();

      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

Пример Fahrenheit

- Такой пример как `PushCounter`, графический интерфейс настраивается в отдельном классе панели
- Внутренний класс `TempListener` определяет слушателя для события создаваемого при действиях в текстовом поле
- Конструктор `FahrenheitPanel` конструктор инициализирует слушателя и добавляет его в текстовое поле
- Когда пользователь задает температуру и нажимает ввод (enter), текстовое поле генерирует событие и производит действия и вызывает `actionPerformed` метод слушателя `actionPerformed`(), который выполняет преобразования и обновляет надпись (label) результата

События клавиатуры

- Событие клавиатуры генерируется, когда пользователь печатает на клавиатуре

Клавиша нажата	***клавиша на клавиатуре нажата***
Клавиша отпущена	***клавиша на клавиатуре высвобождается***
Клавиша напечатана	***клавиша на клавиатуре нажата и освобожден***

- Слушатели для событий клавиатуры создаются, реализуя интерфейс `KeyListener`
- Объект `KeyEvent` передается в соответствующий метод, когда происходит событие, связанное с клавиатурой

События клавиатуры

- Компонент, который генерирует событие нажатия клавиши является тем, который имеет текущий фокус клавиатуры
- `Константы в классе KeyEvent` может быть использован, чтобы определить, какая клавиша была нажата
- Класс KeyEvent содержит большой набор констант. Каждая константа содержит код соответствующей клавиши (нет необходимости знать коды всех клавиш). Достаточно использовать какую-то из констант.
- По названиям констант можно легко определить, какой клавише она соответствует. Например KeyEvent. VK_ENTER или KeyEvent.VK_F.

События клавиатуры

```
JTextField textField = new JTextField(20);
textField.addKeyListener (new KeyListener() {
    public void keyPressed(KeyEvent e) { }
    public void keyReleased(KeyEvent e) { }
    public void keyTyped(KeyEvent e) { }
     } );
```

Если нет необходимости реализовывать все методы KeyListener, то можно сделать вот так:

```
textField.addKeyListener(new KeyAdapter() {

    public void keyPressed(KeyEvent e) {
          }
   });
```

События клавиатуры

Что происходит когда пользователь нажимает клавишу?

- Каждый раз, когда пользователь нажимает клавиши на клавиатуре, то система вызывает методы keyTyped, keyPressed и keyReleased,
- в качестве параметра им передается объект KeyEvent, который содержит всю необходимую информацию о произошедшем событии.

Замечание: поэтому можно узнать код клавиши, которая была нажата, за это отвечает метод getKeyCode.

Например, можно узнат, были ли зажаты при этом такие клавиши, как Alt, Shift или Ctrl. Проверить это можно вызвав соответственно методы isAltDown, isShiftDown и isControlDown.

Помните: события от клавиатуры будут генерироваться системой только тогда, когда компонент, который мы слушаем, находится в фокусе

Пример События клавиатуры

```
import java.awt.Font;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import java.awt.Dimension;
public class TestFrame extends JFrame {


    private JLabel label;


    public TestFrame() {
        super("Test frame");
        createGUI();
    }
```

Пример События клавиатуры

```
public void createGUI() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());
        panel.setFocusable(true);

        label = new JLabel();
        label.setFont(new Font("Calibri", Font.PLAIN, 20));
        label.setHorizontalAlignment(JLabel.CENTER);

        panel.addKeyListener(new KeyAdapter() {
            public void keyReleased(KeyEvent e) {
                label.setText(e.getKeyText(e.getKeyCode()));
            }
        });
```

Пример События клавиатуры

```
panel.add(label, BorderLayout.CENTER);
        setPreferredSize(new Dimension(200, 200));
        getContentPane().add(panel);
    }
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                JFrame.setDefaultLookAndFeelDecorated(true);
                TestFrame frame = new TestFrame();
                frame.pack();
                frame.setLocationRelativeTo(null);
                frame.setVisible(true);
            }
        }); //Обратите внимание!
    }}
```

В Java многопоточность программы организуется с помощью интерфейса **Runnable** и класса Thread, который наследуется от Runnable.
Первый способ более гибкий, второй – проще.

Та часть кода, которая должна выполняться в отдельном потоке, выносится в свой класс, имеющий переопределенный метод run().

Пример События клавиатуры

**Здесь происходит обработки нажатия клавиш компонентом JPanel.
По умолчанию JPanel не должен получать фокуса, однако это можно сделать, если очень захотеть при помощи метода setFocusable и передать этому методу true в качестве параметра.**

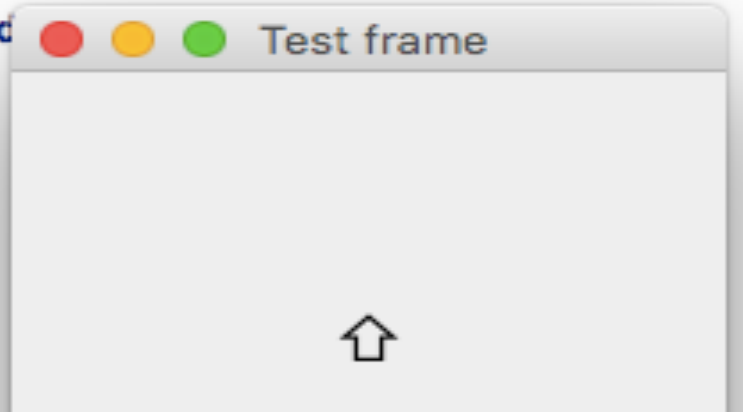
```
import java.awt.event.KeyEvent;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import java.awt.Dimension;


public class TestFrame extend

    private JLabel label;

    public TestFrame() {
        super("Test frame");
        createGUI();
    }
```



События Мыши

- События, связанные с мышью разделены на события мыши и события движения мыши
- События мыши

Нажатие Мыши	***кнопка мыши нажата***
Мышь отпущена	***Кнопка мыши отпущена***
Клик на мышь	***кнопка мыши нажата и освобожден без перемещения мыши между этими событиями***
Мышь вошла	***указатель мыши перемещается на (над) компонент***
Мышь вышла	***указатель мыши перемещается за пределы компонента***

События Мыши

- События движения мыши:

Движение Мыши	мышь перемещается
Перетаскивание мышью	мышь перемещается, в то время пока нажата кнопка мыши

- Слушатели событий мыши создаются при помощи интерфейсов `MouseListener` и `MouseMotionListener`
- Объект `MouseEvent` передается в соответствующий метод, когда происходит событие мыши

События Мыши

- Для данной программы, мы можем обработать только одно или два события мыши
- Для того, чтобы полностью реализовать интерфейс слушателя, пустые методы должны быть предусмотрены для всех неиспользуемых событий

События Мыши

- *Растягивание (Rubberbanding*) это визуальный эффект, в котором форма "растягивается” с помощью мыши

Интерфейс Iterator

- Итератор создается формально, реализовав интерфейс `Iterator`, который содержит три метода
- Метод `hasNext` возвращает логический результат - истинно, если есть элементы которые остались для обработки
- Метод `next` метод возвращает следующий объект в итерации
- Метод `remove` удаляет объект, который совсем недавно, возвратил `next`

Интерфейс Iterator

- Реализуя интерфейс `Iterator`, а класс формально устанавливает, что объекты этого типа являются итераторы
- Программист должен решить, как наилучшим образом реализовать функции итератора
- После того, как появилась для версия for-each для цикла можно использовать для обработки элементов с помощью итераторов

Интерфейсы

- Вы могли бы написать класс, который реализует определенные методы (такой как `compareTo`) без формальной реализации интерфейса (`Comparable`)
- Тем не менее, формально, установление взаимосвязи между классом и интерфейсом позволяет, которые позволяет Java установить связи с объектом в некоторых отношениях

Интерфейсы являются одним из ключевых аспектов объектно-ориентированного проектирования в Java!

Класс ArrayList

- **ArrayList в Java** представляет собой изменяемый список объектов.
- Мы можем добавлять, удалять, находить, сортировать и заменять элементы в этом списке.

Пример с ArrayList

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
public class DotsPanel extends JPanel{
   private final int WIDTH = 300, HEIGHT = 200;
   private final int RADIUS = 6;
   private ArrayList pointList;
   private int count;
//Устанавливаем эту панель, чтобы слушать события мыши.
public DotsPanel() {  //конструктор класса
      pointList = new ArrayList();
      count = 0;
      addMouseListener (new DotsListener());
      setBackground (Color.black);
      setPreferredSize (new Dimension(WIDTH, HEIGHT)); }
```

```
//рисуем все точки, которые хранятся в списке
public void paintComponent (Graphics page)  {
      super.paintComponent(page);
      page.setColor (Color.green);
      // создаем  итератор для ArrayList точек
Iterator pointIterator = pointList.iterator();
      while (pointIterator.hasNext())  {
         Point drawPoint = (Point) pointIterator.next();
         page.fillOval (drawPoint.x - RADIUS, drawPoint.y -
RADIUS,
                        RADIUS * 2, RADIUS * 2);
      }
      page.drawString ("Count: " + count, 5, 15);
   }
```

```
//класс слушателя событий мыши.
//*****************************************************
*
   private class DotsListener implements MouseListener {
//  добавляет текущую точку в список точек и перерисовываем
      // При каждом нажатии кнопки мыши.
public void mousePressed (MouseEvent event)  {
         pointList.add (event.getPoint());
         count++;
         repaint();
//  Обеспечить пустые определения для неиспользуемых методов
событий.
      public void mouseClicked (MouseEvent event) {}
      public void mouseReleased (MouseEvent event) {}
      public void mouseEntered (MouseEvent event) {}
      public void mouseExited (MouseEvent event) {}
   }
}
```

Пример ArrayList

```
import javax.swing.JFrame;


public class Dots {
    public static void main (String[] args)
    {
        JFrame frame = new JFrame ("Dots");
        frame.setDefaultCloseOperation
(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add (new DotsPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
```

Диалоговые окна

- Напомним, что диалоговое окно это небольшое окно, которое "всплывает", чтобы взаимодействовать с пользователем в течение краткого время, для достижения конкретной цели
- Класс `JOptionPane` позволяет легко создавать диалоговые окна для представления информации, подтверждающей какие-то действия, или чтобы принимать некоторые значения ввода
- Давайте теперь рассмотрим использование диалоговых окон на двух других классах, которые позволяют нам создавать специализированные диалоговые окна

Исключения

- Обработка исключений является важным аспектом объектно-ориентированного проектирования
- Мы рассмотрим:
  - понятие исключения
  - сообщения исключений
  - использование оператора try catch
  - распространяющихся исключений
  - иерархия классов исключений

Исключения в Java

Это особый механизм, который позволяет защитить ваш код от ошибок

Исключения

Понятие исключения или исключительной ситуации

- Исключение это объект, который описывает необычное или ошибочное поведение программы
- Исключения могут быть выброшены в одной части программы, и затем могут быть перехвачены и обработаны с помощью другой ее части
- Программа может быть разделена на нормальный поток выполнения и поток выполнения исключений
- В Java ошибка также представлена как объект, но, как правило, ошибка это некий участок кода непокрытый тестами, ее нельзя отловить, за счет которого происходит неправильное поведение программы,

Исключения в Java

Представляют особый механизм, который позволяет защитить ваш код от ошибок

```
try{
.....
// блок операторов который выполняем
      } catch(Класс_исключения имя){
…
// сюда попадаем, если что-то пошло не так
}
```

Исключения Java

```
int a = 5;
int b = 0;
try {
        float c = a / b;
} catch (ArithmeticException e) {
    System.out.println("Делить на ноль нельзя!");
}
```

Обработка исключений в Java

- Java имеет заранее определенный набор исключений и ошибок, которые могут возникнуть во время выполнения
- Программа может иметь дело с исключением в одном из трех способов:
  - *Игнорировать их*
  - *Обработка, там где исключение произошла*
  - *Обработка в другом месте*
- Способ, в котором обрабатывается исключение является важным фактором проектирования программ

Полезные ссылки

- https://younglinux.info/java/runnable
- https://docs.oracle.com/javase/tutorial/uiswing/concurrency/initial.html
- Swing articles: http://java.sun.com/javase/technologies/desktop/articles.jsp
- Swing Architecture: http://java.sun.com/products/jfc/tsc/articles/architecture/
- Visual Editor for Eclipse: http://wiki.eclipse.org/VE/Update#Online_Install
- Oracle Swing tutorial: http://download.oracle.com/javase/tutorial/uiswing/
- Stack Overflow: http://stackoverflow.com/

Полезные ссылки

- Basic Swing tutorial: http://zetcode.com/tutorials/javaswingtutorial/
- *http://zetcode.com/tutorials/javaswingtutorial/swinglayoutmanagement/*
- *http://www.macs.hw.ac.uk/cs/java-swing-guidebook/?name=Layouts&page=3*
- *http://www.quizful.net/post/swing-layout-managers*
- *https://javaswing.wordpress.com/2009/12/23/keylistener_using/*

Спасибо за внимание!