ГРАФИЧЕН ПОТРЕБИТЕЛСКИ ИНТЕРФЕЙС

гл.ас. д-р Мария Евтимова

https://github.com/marias83837/JavaPresentations

Приложения с GUI

- Създават прозорци върху екрана
- Съдържат визуални елементи за:

-изобразяване на информация

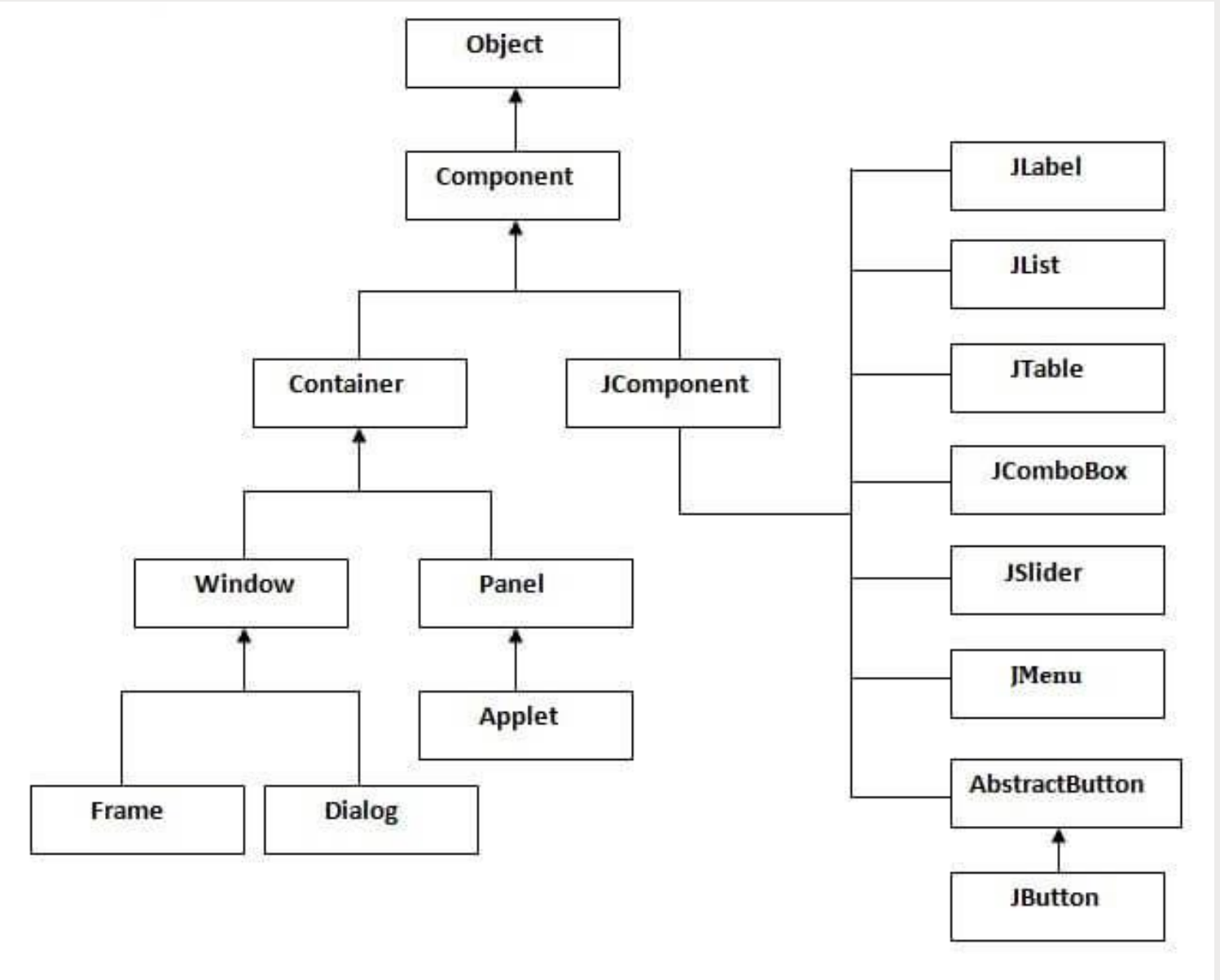
-взаимодействие на потребителя с програмата чрез мишка или клавиатура

- Класове от пакетите java.awt или javax.swing

Предимства на Swing

- платформено независим
- приспособим
- разширяем
- конфигуруем
- по- лек

Йерархия на класовете



Компонента

- Обект, който представя графична информация или позволява на потребителя да взаимодейства с програмата по някакъв начин
- Примери: бутони, текстови полета, етикети, плъзгачи, менюта
- Генерира събития
- Програмите отговарят на събитията чрез слушателски класове
- Всяко събитие има съответстващ адаптерен клас
- Подходящият слушател се добавя към компонентата, генерираща събитието

Клас Component (java.awt)
Супер клас за всички графични компоненти

- public void paint (Graphics page)

Изчертава компонентата с графично съдържание page

- public void repaint()

Пречертава компонентата

- public void update(Graphics page)

Обновява компонентата, използвайки графичното съдържание page

- public Color getBackground()
- public Color getForeground()
- public void setBackground(Color color)
- public void setForeground(Color color)

Определя цвета на фона или цвета за изчертаване на компонентата

- **public Dimension getSize()**
- **public void setSize(Dimension dim)**
- **public void setSize(int width, int height)**

Определя размера на компонентата

- **public void setLocation(int x, int y)**

Премества компонентата в ново местоположение с горен ляв ъгъл, определен от x и y параметрите в координатното пространство на компонентата- родител

- **public synchronized void add(PopupMenu popmenu)**
- **public synchronized void remove(PopupMenu popmenu)**

Добавя/премахва издигащото се меню popmenu към компонентата

- **public Graphics getGraphics()**

Връща графичното съдържание на компонентата

public void setVisible(boolean b)

Изобразява/скрива компонентата според стойността на b(true/false)

Събития с компоненти

- ComponentEvent
- FocusEvent
- KeyEvent
- MouseEvent
- MouseMotionEvent
- MouseWheelEvent

Добавяне на слушатели на събитията

public void addXXListener(XXListener I)

XX- **име на събитие**

Клас Jcomponent

- Базов клас на всички Swing компоненти
- Без контейнерите на по- високо ниво

Контейнер

- Компонента, която може да съдържа други компоненти
- Контейнерите и свързаните с тях мениджъри на разположението определят организирането и изобразяването на компонентите (интерфейс LayoutManager)
- Видове контейнери

- Аплет
- Фрейм- прозорец, който може да се разположи на произволно място върху екрана
- Панел- организира групи от компоненти в по- голям контейнер; не може да бъде изобразяван и трябва да се добави към съществуващ контейнер

Фрейми и аплети- контейнери от горно ниво

Клас Container (java.awt)
Супер клас за контейнери

- **public Component add(Component item)**
- **public Component add(Component item, int index)**
- **public Component add(String str, Component item)**

Добавя компонента **item** към контейнера в определено място **index** (към края на подразбиране) с име **str**

- **public Component getComponent(int index)**
- **public Component getComponent(int x, int y)**
- **public Component getComponent(Point p)**

Връща компонента, разположена в определена точка **point** или **index**

- **public LayoutManager getLayout()**
- **public void setLayout(LayoutManager layout)**

Връща/ установява мениджъра на разположение за контейнера

Клас javax.swing.JFrame
Представя прозорец

- **public JFrame(String title)**

Конструкторът създава прозорец със заглавие **title**

- **public void setDefaultCloseOperation(int operation)**

Установява операцията при затваряне на прозореца

-JFrame.DO_NOTHING_ON_CLOSE- нищо

-JFrame.HIDE_ON_CLOSE – скрий прозореца

-JFrame.DISPOSE_ON_CLOSE-освободи прозореца

-JFrame.EXIT_ON_CLOSE-завърши приложението

Javax.swing.JFrame

- **public Container getContentPane()**

Връща обект от класа Container със съдържанието на фрейма

- **public void setContentPane(Container contentPane)**

Установява съдържанието на фрейма

Събития

- Всички събития за компоненти
- Събития с прозорци

- клас **WindowEvent**
- Интерфейс **WindowListener** или адаптерен клас **WindowAdapter**
- добавяне на слушател /за събитието **WindowsEvent** към компонентата

public void addWindowListener(WindowListener I)

- **public interface WindowListener{**

//Извиква се, когато прозорецът се установи като активен прозорец, т.е. ще получава събития от клавиатурата

- **void windowActivated(WindowEvent e);**

//Извиква се, когато прозорецът се затвори при разрушаването му

- **void windowClosed(WindowEvent e);**

//Извиква се, когато прозорецът се затвори от системното си меню

- **void windowClosing(WindowEvent e);**

//Извиква се, когато прозорецът повече не е активен, т.е. престава да получава събития от клавиатурата

- **void windowDeactivated(WindowEvent e);**

}

public interface WindowListener{

Извиква се при промяна на прозореца от икона в нормално състояние

- **void windowDeiconified(WindowEvent e);**

//Извиква се при промяна на прозореца от нормално състояние в икона

- **void windowIconified(WindowEvent e);**

//Извиква се, когато прозоцът е станал видим за първи път

- **void windowOpened(WindowEvent e);**

}

Клас **javax.swing.SwingUtilities**

Съдържа помощни методи за Swing

или

Клас **java.awt.EventQueue**

Независим от платформата клас, който обработва събитията;

public static void invokeLater(Runnable doRun)

Предизвиква асинхронно изпълнение на метода **doRun.run()** в нишката (клас **Thread**, реализиращ интерфейса **Runnable**), изпълняваща събитието

Клас javax.swing.JOptionPane

Представя стандартен диалогов прозорец за:

- потвърждение **showConfirmDialog**
- входен диалог **showInputDialog**
- за съобщение **showMessageDialog**
- за диалог с опции **showOptionDialog**

public static void showMessageDialog(Component parentComponent, Object message, String title, int message Type, Icon icon)

parentComponent- компонента- родител

message- съобщение, което се изобразява

title- заглавие на диалоговия прозорец

Клас javax.swing.JOptionPane

messageType- тип на съобщението

ERROR_MESSAGE

INFORMATION_MESSAGE

WARNING_MESSAGE

QUESTION_MESSAGE

QUESTION_MESSAGE

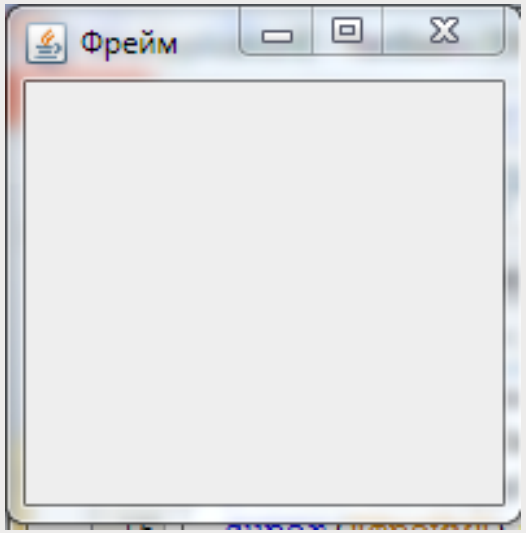
PLAIN_MESSAGE

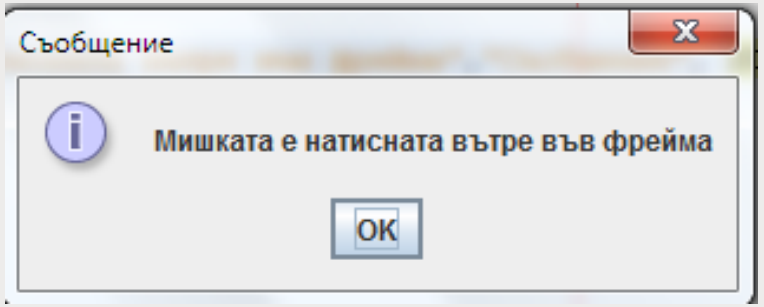
icon- изобразявана икона

Приложение: Приложение, което обработва събитието натискане бутон на мишката- изобразява се диалогов прозорец със съобщението

```
import  java.awt.event.*;
import java.awt.*;
import javax.swing.*;
public class JavaApplicationE extends JFrame{
public JavaApplicationE(){
super("Фрейм");
//Извиква конструктора на суперкласа JFrame за създаване на прозорец със заглавие Фрейм
//Приложението завършва при затваряне на прозореца
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//Добавя слушател за събитието “натискане на мишката”
 this.addMouseListener(new MouseAdapter(){
//Обработва събитието “Натискане на мишката”
public void mousePressed(MouseEvent e){
JOptionPane.showMessageDialog(null, "Мишката е натисната вътре във фрейма","Съобщение", JOptionPane.INFORMATION_MESSAGE);
}
});
//Установява размера на прозореца
this.setSize(200,200);
}
```

```
public static void main(String[] args){
//Създава и изпълнява формата
EventQueue.invokeLater(new Runnable(){
@Override
public void run(){
new JavaApplicationE().setVisible(true);
}
        });
}
}
```





Панел

Клас **javax.swing.JPanel**

Представя контейнер

public JPanel()

Създава контейнер

public void setBorder(Border border)

Установява рамка на панела

public Component add(Component comp)

Добавя дадената компонента comp към контейнера

Jpanel panel;

panel= new Jpanel();

panel.setBorder(new TitledBorder(new EtchedBorder(), ”Име”));

Бутон

Клас **javax.swing.JButton**

Създава бутон. При натискането му се генерира събитието ActionEvent, изпраща се съобщението actionPerformed на интерфейса ActionListener за всички слушатели, които се регистрират чрез метода addActionListener

public JButton()

Конструира JButton без етикет

public JButton(String label)

Конструира JButton с определен етикет label

public void addActionListener(ActionListener I)

Добавя слушател I, за да приеме събитията ActionEvent при натискане на бутона. Методът getSource на събитието ActionEvent

public Object getSource()

връща обекта- източник на събитието

public interface ActionListener{

//Извиква се, когато се случи събитието **ActionEvent**, показва източника на събитието и потребителят описва действието, което трябва да се предприеме

void actionPerformed(ActionEvent e) ;

}

Етикет

Клас **javax.swing.JLabel**

Изобразява област за къс текст, изображение или и двете. Не реагира на входни събития.

public JLabel(String text)

Създава етикет с определен текст **text**

Текстови компоненти

Клас **javax.swing.JTextField-** текстово поле

JTextField се използва за текст от един ред. При натискане на **<Enter>** текстовите контроли вдигат събитието **ActionEvent**, изпраща се съобщението **actionPerformed** на интерфейса **ActionListener** за всички слушатели, които са се регистрирали чрез метода **addActionListener.**

public JTextField(Document doc, String text, int columns)

text- начален текст (по подразбиране null);

columns-брой колони (по подразбиране 0);

doc- определен модел за запазване на текста (null)

public void addActionListener(ActionListener I)

Добавя слушател **I** на събитието **ActionEvent** към полето.

public String getText()

public void setText(String t)

Връща/установява текста **t** в текстовото поле.

Клас javax.swing.JTextArea-текстова област

JTextArea се използва за текст от много редове. За да използва плъзгачи, **JTextArea** се разполага в компонента **JScrollPane.** Промяна на текста се предава от модела за запазване на текста чрез събитието **DocumentEvent,** изпращат се съобщенията **changeUpdate, insertUpdate, removeUpdate** на интерфейса **DocumentListener** за всички слушатели, които са се регистрирали.

public JTextArea(String text, int rows, int columns)

public JTextArea(Document doc)

text- начален текст (по подразбиране null)

rows- брой редове(по подразбиране null)

columns- брой колони(по подразбиране 0)

doc- определен модел за запазване на текста (по подразбиране null)

public void append(String str)

Добавя даден текст **str** към края на документа.

Клас javax.swing.JScrollPane

Представя плъзгач за компонента.

public JScrollPane(Component view, int vsbPolicy, int hsbPolicy)

view- компонента, която ще използва плъзгач;

vsbPolicy- определя вида на вертикалния плъзгач:

VERTICAL_SCROLLBAR_ALWAYS
VERTICAL_SCROLLBAR_AS_NEEDED

VERTICAL_SCROLLBAR_NEVER

hsbPolicy – определя вида на хоризонталния плъзгач:

HORIZONTAL_SCROLLBAR_ALWAYS

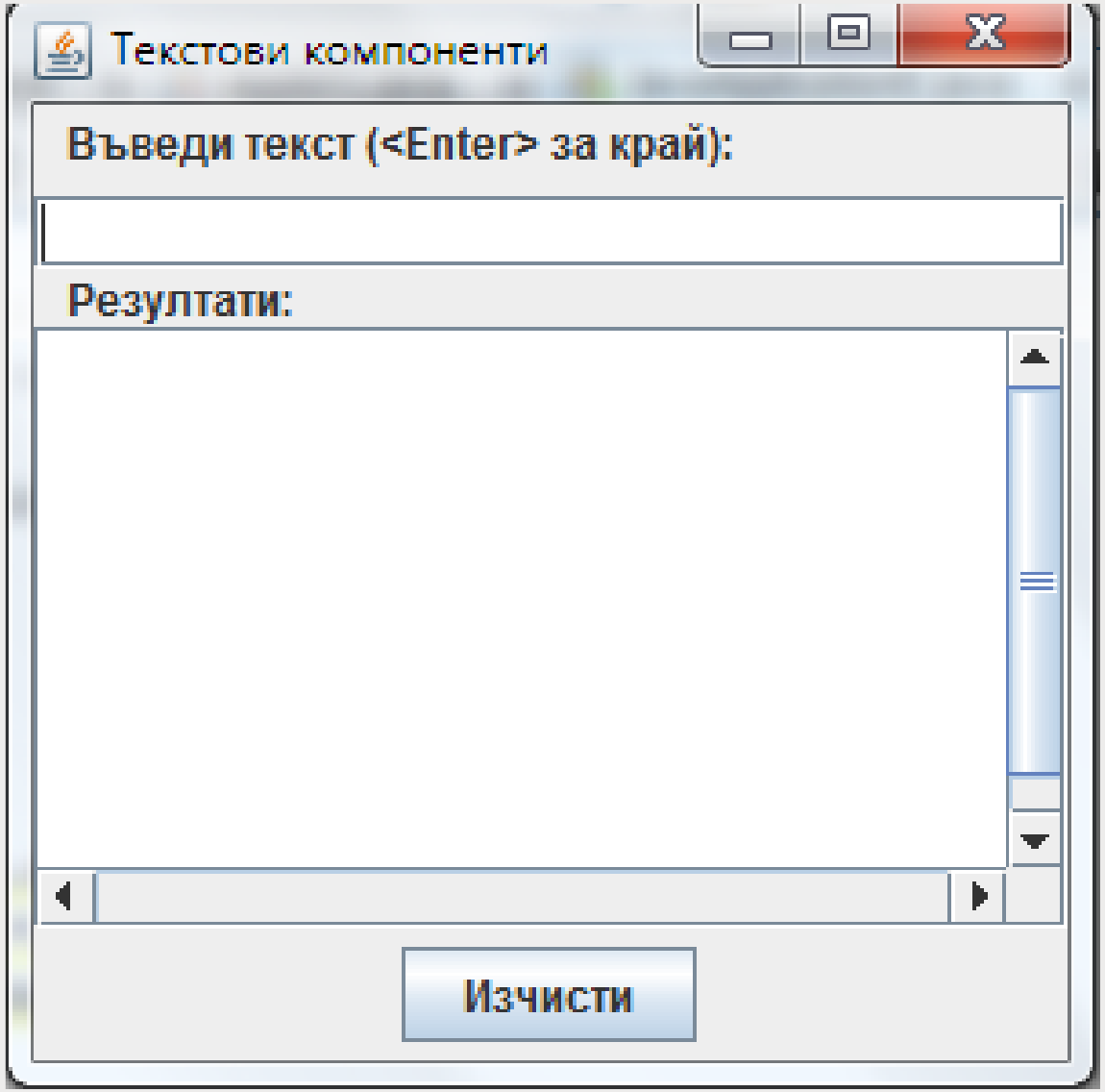
HORIZONTAL_SCROLLBAR_AS_NEEDED

HORIZONTAL_SCROLLBAR_NEVER

Пример: Приложение, което въвежда текст в текстово поле и при натискане на **<Enter>** въведеният текст се добавя в текстовата област. При натискане на бутона **Изчисти** се изчиства целият текст от текстовата област. Използва събитието **ActionEvent** за текстовото поле и за бутона.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class JavaApplicationE extends JFrame implements ActionListener {
private JPanel panel1, panel2, panel3;
private JLabel label1, label2;
private JTextField field;
private JTextArea area;
private JScrollPane scroll;
private JButton clear;
public JavaApplicationE () {
super("Текстови компоненти");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
Container container = this.getContentPane();
panel1 = new JPanel();// панел с етикет и текстово поле
panel1.setLayout(new GridLayout(2, 1, 5, 5));
label1 = new JLabel("   Въведи текст (<Enter> за край):   ");
panel1.add(label1);
field = new JTextField(20);
field.addActionListener(this);
panel1.add(field);
container.add(panel1, BorderLayout.PAGE_START);
panel2 =new JPanel();// панел с етикет и текстова област
panel2.setLayout(new BorderLayout());
label2 = new JLabel("   Резултати:   ");
panel2.add(label2, BorderLayout.PAGE_START);
area = new JTextArea(10, 20);
scroll = new JScrollPane(area,
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
panel2.add(scroll, BorderLayout.CENTER);
container.add(panel2, BorderLayout.CENTER);
panel3 = new JPanel();//панел с бутон
clear = new JButton("Изчисти");
clear.addActionListener(this);
panel3.add(clear);
container.add(panel3, BorderLayout.PAGE_END);
this.setSize(300, 300);}
```

```
@Override
public void actionPerformed(ActionEvent e) {
Object source = e.getSource();
if(source == field){
// Полето е източник на събитието
String input = field.getText();
// взима въведения текст във field
area.append(input + "\n");
// и го добавя в area
field.setText("");
// изчиства field
}else if(source == clear)
// Бутонът е източник на събитието
area.setText("");// изчиства area
}
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
@Override
public void run() {
new JavaApplicationE().setVisible(true);
}});
}
```


Текстови компоненти
Въведи текст (<Enter> за край):
Резултати:
Изчисти

Компоненти за избор

- Бутон за избор- клас **javax.swing.JCheckBox**
- Радио бутон- клас **javax.swing.JRadioButton**
- Комбиниран бутон за избор- клас **javax.swing.JComboBox**
- Списък- клас **javax.swing.JList**
- Модел за елементите на динамичен списък- клас **javax.swing.DefaultListModel**

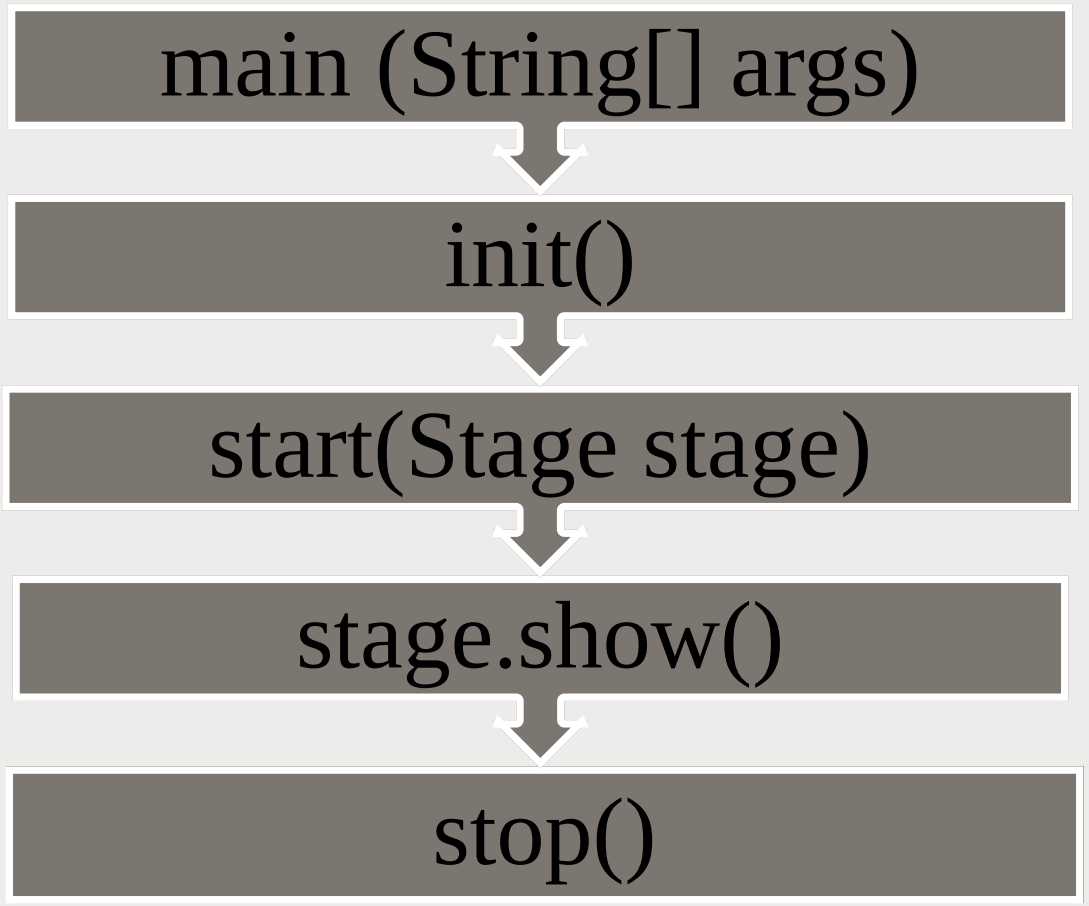
Меню

1. Меню лента- клас **javax.swing.JMenuBar**
2. Меню- клас **javax.swing.Jmenu**
3. Меню елемент- клас **javax.swing.JMenuItem**
4. Меню елемент с избор- клас **javax.swing.JCheckBoxMenuItem**
5. Радио бутон меню елемент- клас **javax.swing.JRadioButtonMenuItem**

JavaFx- е мощен графичен и медиен фреймуърк

- **SceneBuilder-** GUI дазайн инструмент, който позволява на разработчика да създаде потребителски интерфейс визуално, без да е необходимо да се отделя време за позициониране на компонентите върху интерфейса
- **CSS**(Cascading Style Sheet)- език за стилове, който описва презентацията на документите и най- често се асоциира с HTML документите в интернет
- **FXML-** (JAVA XML език) разширяем маркиращ език за Java. Указва само как да бъде структуриран един документ

Жизнен цикъл на JavaFX приложение

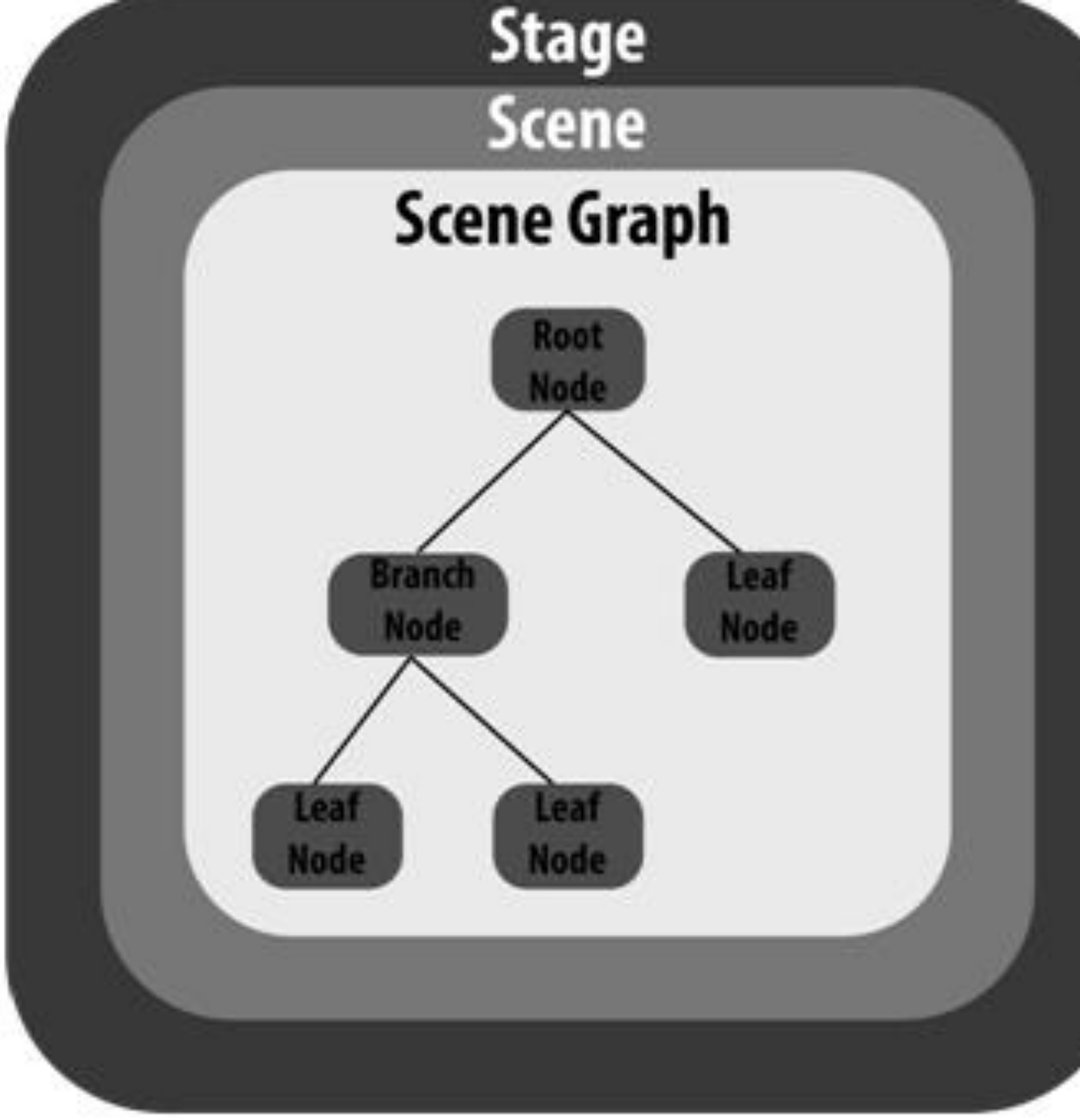


Стъпки на изпълнение на JavaFx приложение

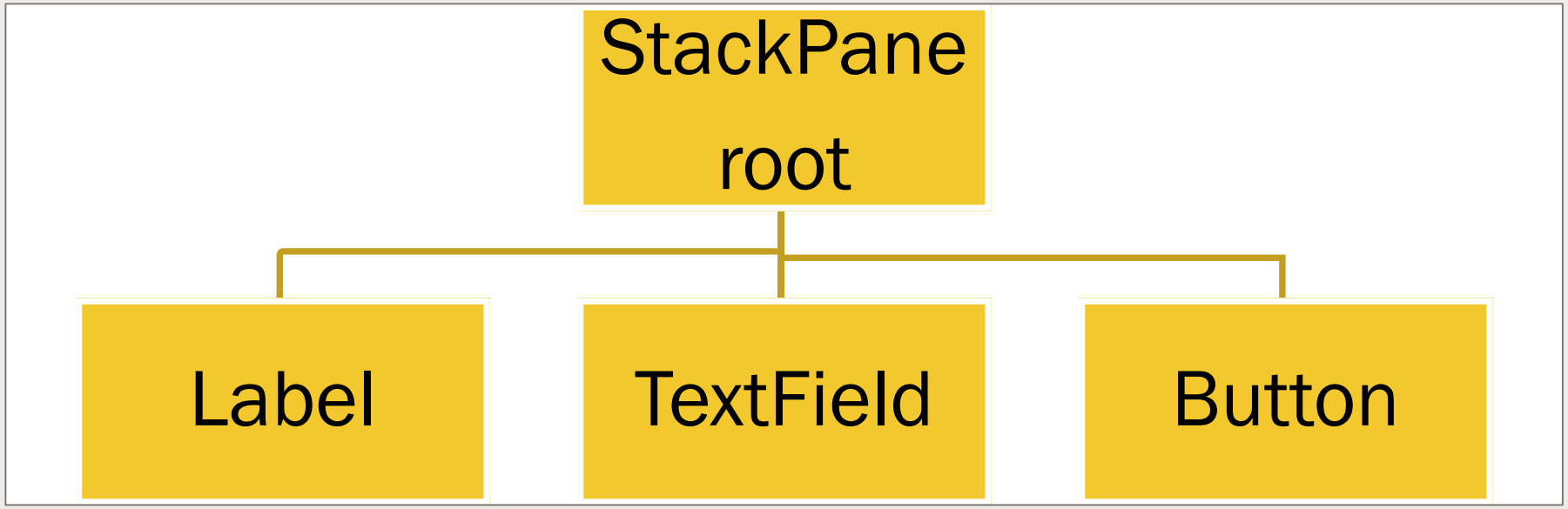
1. **JavaFx** конструира инстанция на определения клас (Application)
2. **init()** метода се изпълнява
3. **start()** метода се изпълнява
4. **JAVAFX** чака приложението да свърши, което се осъществява при изпълнение на метода **Platform.exit()** или когато последния прозорец на приложението се затвори и след това се извиква **stop().**

```
import  javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class LifeCycleExample extends Application{
public static void main(String [] args)
{
launch(args) ;
}
@Override
public void init()
{
System.out.println("В метода init()");
}
@Override
public void start(Stage primaryStage)
{
primaryStage.setTitle("Жизнен цикъл");
StackPane root= new StackPane();
primaryStage.setScene(new Scene(root,300,75));
primaryStage.show();
System.out.println("В метода start()");
}
@Override
public void stop(){
System.out.println("В метода stop()");
}
}
```

JAVAFX структура

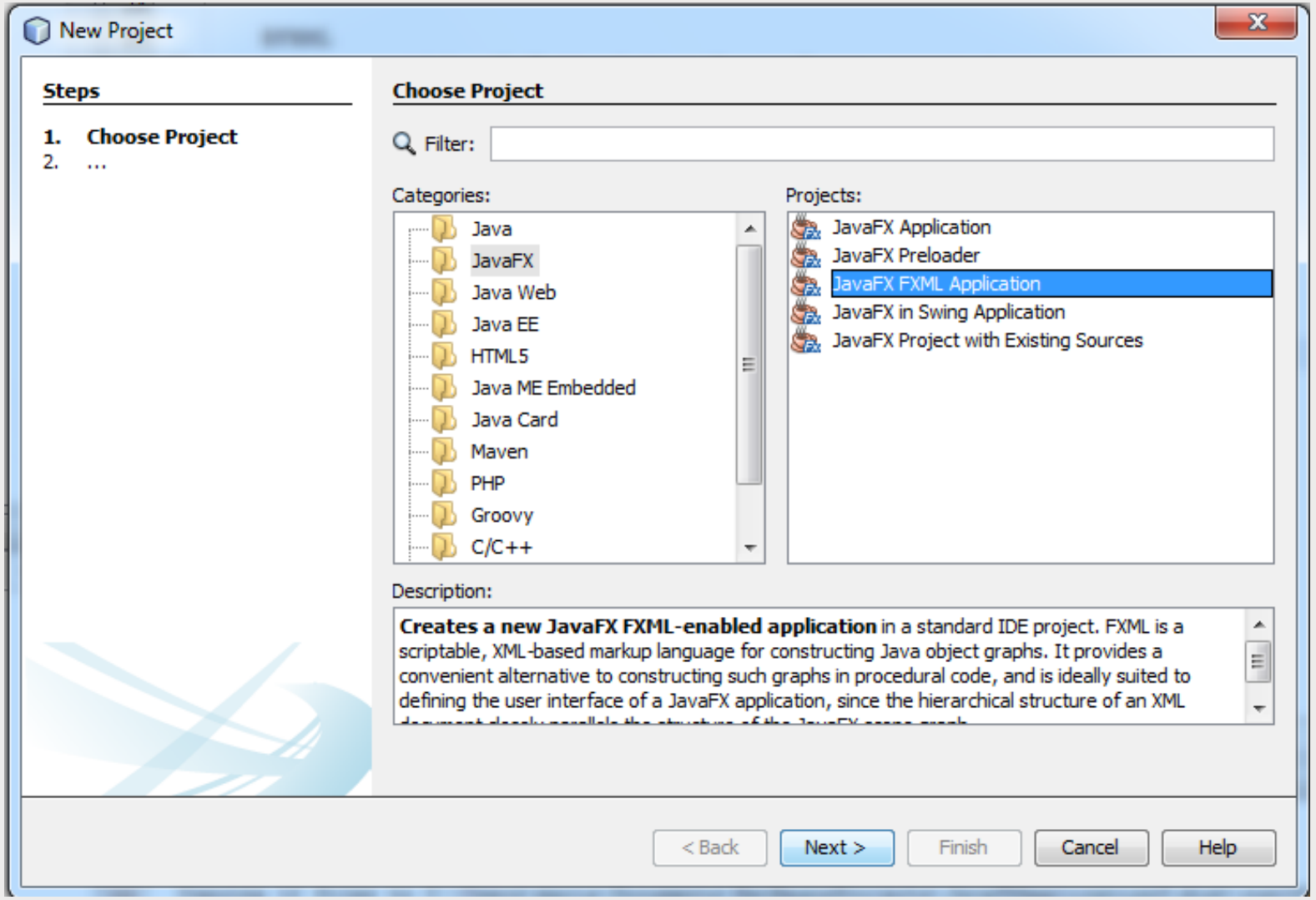


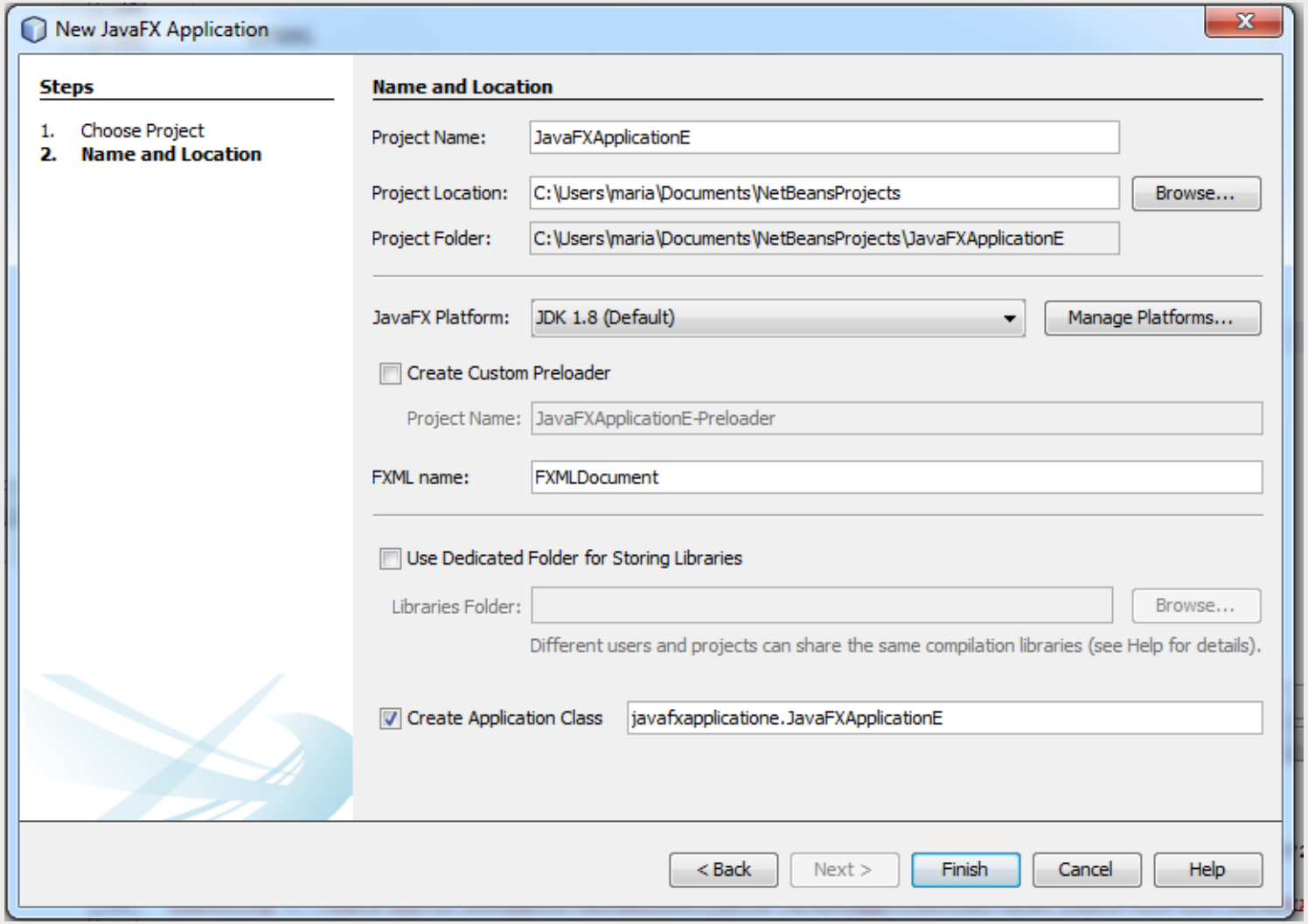
- **javafx.stage.Stage** клас- представя целия прозорец на приложението
- **javafx.scene.Scene** клас- приема съдържанието вътре в прозореца
- **javafx.scene.layout.StackPane** - определя разположението на дизайна на сцената

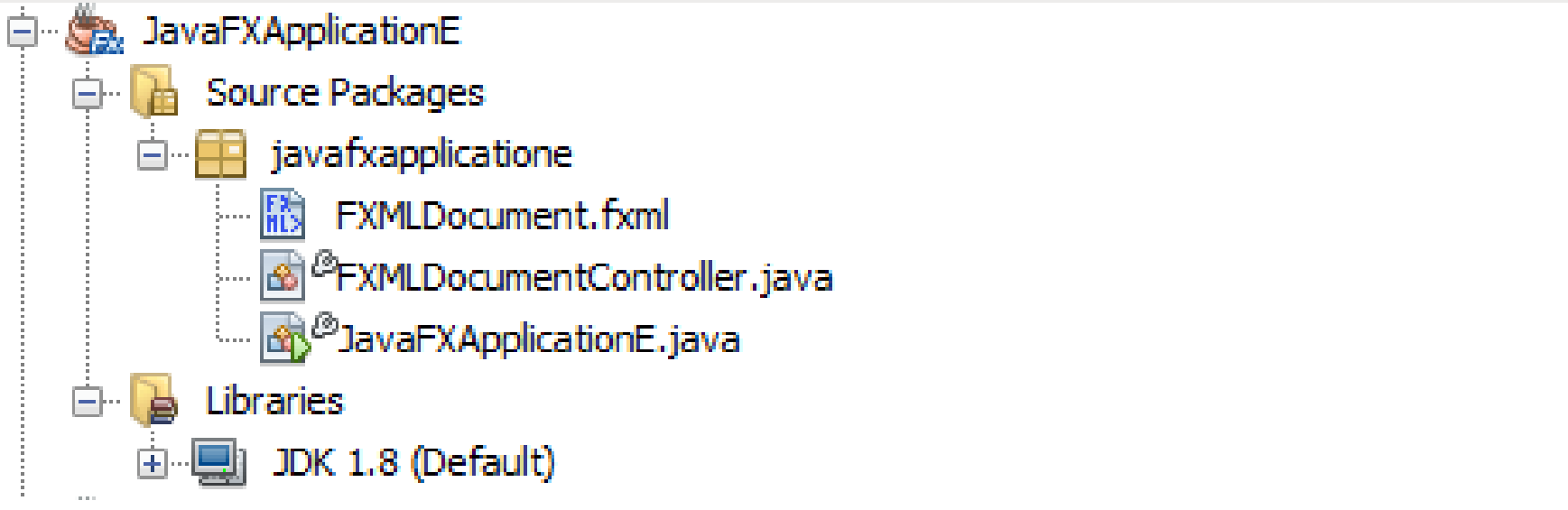


Пример:

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import  javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class JavaFXExample extends Application{
@Override
 public void  start(Stage primaryStage)
{
Button btn= new Button();
btn.setText("Confirm");
btn.setOnAction(new EventHandler<ActionEvent>()
{
@Override
public void handle(ActionEvent event)
{
System.out.println("JavaFX пример");
}});
StackPane root=new StackPane();
root.getChildren().add(btn);
Scene scene= new Scene(root,400,300);
primaryStage.setTitle("Пример на JavaFX");
primaryStage.setScene(scene);
primaryStage.show();
}
```


New Project
Steps
1. Choose Project
2. ...
Choose Project
Filter:
Categories:
Java
JavaFX
Java Web
Java EE
HTML5
Java ME Embedded
Java Card
Maven
PHP
Groovy
C/C++
Projects:
JavaFX Application
JavaFX Preloader
JavaFX FXML Application
JavaFX in Swing Application
JavaFX Project with Existing Sources
Description:
Creates a new JavaFX FXML-enabled application in a standard IDE project. FXML is a scriptable, XML-based markup language for constructing Java object graphs. It provides a convenient alternative to constructing such graphs in procedural code, and is ideally suited to defining the user interface of a JavaFX application, since the hierarchical structure of an XML
< Back
Next >
Finish
Cancel
Help


New JavaFX Application
Steps
1. Choose Project
2. Name and Location
Name and Location
Project Name: JavaFXApplicationE
Project Location: C:\Users\maria\Documents\NetBeansProjects
Browse...
Project Folder: C:\Users\maria\Documents\NetBeansProjects\JavaFXApplicationE
JavaFX Platform: JDK 1.8 (Default)
Manage Platforms...
Create Custom Preloader
Project Name: JavaFXApplicationE-Preloader
FXML name: FXMLDocument
Use Dedicated Folder for Storing Libraries
Libraries Folder:
Browse...
Different users and projects can share the same compilation libraries (see Help for details).
Create Application Class javafxapplicatione.JavaFXApplicationE
< Back
Next >
Finish
Cancel
Help


JavaFXApplicationE
Source Packages
javafxapplicatione
FXMLDocument.fxml
FXMLDocumentController.java
JavaFXApplicationE.java
Libraries
JDK 1.8 (Default)

FXMLDocument.fxml

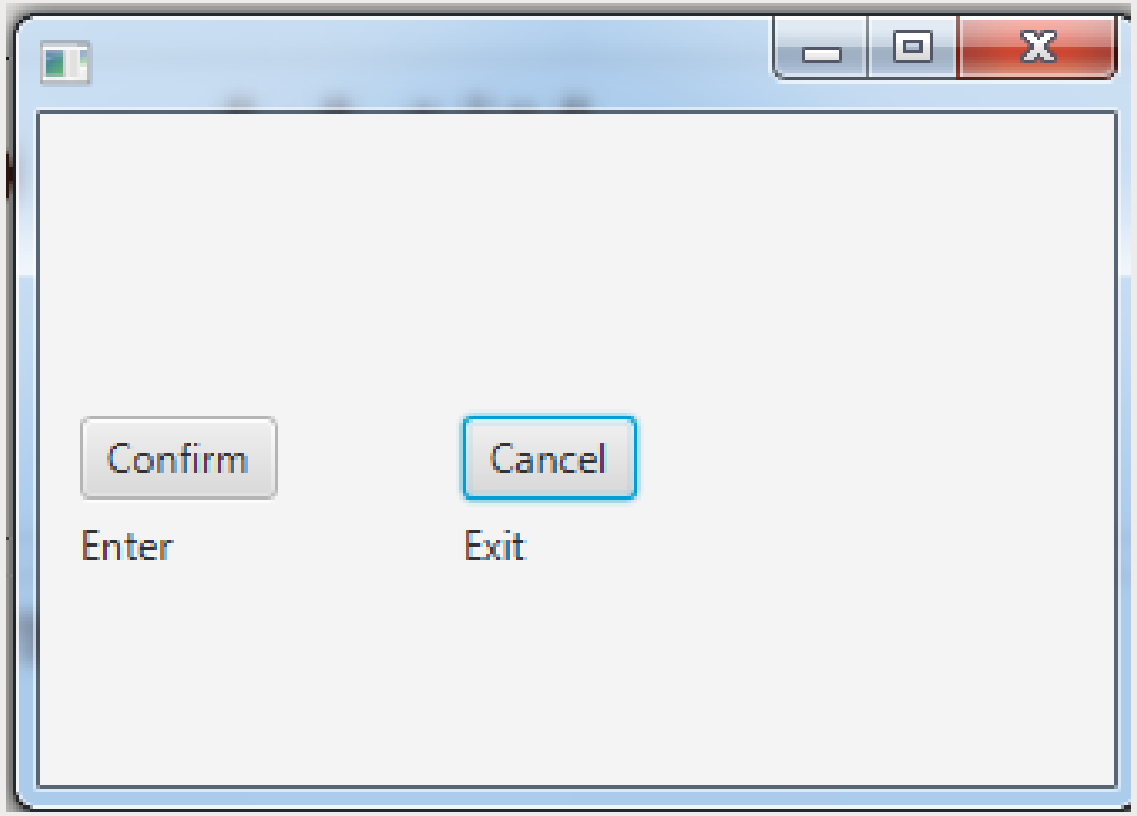
```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>


<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320" xmlns:fx="http://javafx.com/fxml/1" fx:controller="javafxapplication3.FXMLDocumentController">
    <children>
        <Button layoutX="126" layoutY="90" text="Cancel" onAction="#handleButtonAction" fx:id="button" />
        <Label layoutX="126" layoutY="120" minHeight="16" minWidth="69" fx:id="label" />
        <Button layoutX="12" layoutY="90" text="Confirm" onAction="#handleButtonAction1" fx:id="button1" />
        <Label layoutX="12" layoutY="120" minHeight="16" minWidth="69" fx:id="label1" />
    </children>
</AnchorPane>
```

FXMLDocumentController.java

```
import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Label;

/**
 *
 * @author maria
 */
public class FXMLDocumentController implements Initializable {

    @FXML
    private Label label;

    @FXML
    private void handleButtonAction(ActionEvent event) {
        System.out.println("Exit");
        label.setText("Exit");
    }
    @FXML
    private Label label1;

    @FXML
    private void handleButtonAction1(ActionEvent event) {
        System.out.println("Enter");
        label1.setText("Enter");
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

}
```

JavaFXApplicationE.java

```
package javafxapplicatione;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

/**
 *
 * @author maria
 */
public class JavaFXApplicationE extends Application
{

    @Override
    public void start(Stage stage) throws Exception {
        Parent root =
FXMLLoader.load(getClass().getResource("FXMLDo
cument.fxml"));

        Scene scene = new Scene(root);

        stage.setScene(scene);
        stage.show();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```


Confirm
Cancel
Enter
Exit