

# Лекция 14.1

## Основни компоненти на графичния интерфейс (II част)

# Основни теми

- Принципите за моделиране на потребителски графичен интерфейс (GUI).
- Построяване на GUI и обработка на събития породени от събития в резултат от взаимодействие на потребителя с графичната среда.
- Представяне на основните библиотеки с компоненти на графичния интерфейс.
- Създаване и работа с бутони, етикети, текстови полета и панели.
- Обработка на събития, породени от мишка и клавиатура.
- Използване на менажери за управление на разположението на графичните компоненти

- 11.1 Въведение
- 11.2 Графичен интерфейс за вход и изход JOptionPane
- 11.3 Преглед на Swing компоненти
- 11.4 Изобразяване на Text и Image в графичен прозорец
- 11.5 Текстови полета и въведение към обработка на събития с вложени класове**
- 11.6 Общи GUI типове на събития и Listener интерфейс-и**
- 11.7 Схематично представяне на модела за обработка на събития в Java**
- 11.8 JButton компоненти**
- 11.9 Бутони, които задават промяна на състояние
  - 11.9.1 JCheckBox компоненти
  - 11.9.2 JRadioButton компоненти
- 11.10 JComboBox and Using an Anonymous Inner Class for Event Handling

**Литература:**

***Java How to Program, Sixth Edition, глава 11***

11.11 JList КОМПОНЕНТИ

11.12 Multiple-Selection Lists

**11.13** Обработка на събития, породени от мишката

**11.14** Адаптер класове за обработка на събития

**11.15** Обработка на събития, породени от клавиатурата

11.17 Менажери за управление на разположението

11.17.1 FlowLayout

11.17.2 BorderLayout

11.17.3 GridLayout

11.18 Using Panels to Manage More Complex Layouts

**11.19** JTextArea КОМПОНЕНТИ

**Задачи**

**Литература:**

*Java How to Program, Sixth Edition, глава 11*

# 11.5 Текстови полета и въведение към обработка на събитие `ActionEvent`

- Преговор от предишната лекция (15.2)
- `class JTextComponent`
  - Базов клас на `class JTextField`
  - Базов клас на `class JPasswordField`
    - Добавя `echo` символ (звезда) за скриване на текста при печатането му в текстовото поле
  - Позволява на потребителя да въвежда текст, когато компонентата получи фокус.
  - Поражда събитието `ActionEvent`, което се обработва с метод `actionPerformed()`
  - За еднообразие, всеки клиентски клас, който обработва `ActionEvent` трябва да реализира метод `actionPerformed()` на `interface ActionListener`

## Outline

TextFieldFrame  
.java

(1 от 3)

```
1 // Fig. 11.9: TextFieldFrame.java
2 // Demonstrating the JTextField class implementing ActionListener interface.
3 import java.awt.FlowLayout;
4 import java.awt.event.ActionListener;
5 import java.awt.event.ActionEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JTextField;
8 import javax.swing.JPasswordField;
9 import javax.swing.JOptionPane;
10
11 public class TextFieldFrame extends JFrame implements ActionListener
12 {
13     private JTextField textField1; // text field with set size
14     private JTextField textField2; // text field constructed with text
15     private JTextField textField3; // text field with text and size
16     private JPasswordField passwordField; // password field with text
17
18     // TextFieldFrame constructor adds JTextFields to JFrame
19     public TextFieldFrame()
20     {
21         super( "Testing JTextField and JPasswordField" );
22         setLayout( new FlowLayout() ); // set frame layout
23
24         // construct textfield with 10 columns
25         textField1 = new JTextField( 10 );
26         add( textField1 ); // add textField1 to JFrame
27     }
28 }
```

Реализира интерфейс за  
обработка на събитието  
**ActionEvent**

Създава обект **JTextField**



## Outline

TextFieldFrame  
.java

(2 от 3)

Създава обект JTextField

Създава обект JTextField  
без да позволяваСъздава обект  
JPasswordFieldРеализира обработка на  
actionPerformed в текущия  
обект **this** (регистрира  
метод за обработка на  
събитие към Swing  
компонетите)Дефинира метод actionPerformed за обработка на  
събитието

```

28 // construct textfield with default text
29 textField2 = new JTextField( "Enter text here" );
30 add( textField2 ); // add textField2 to JFrame
31
32 // construct textfield with default text and 21 columns
33 textField3 = new JTextField( "Uneditable text field", 21 );
34 textField3.setEditable( false ); // disable editing
35 add( textField3 ); // add textField3 to JFrame
36
37 // construct passwordfield with default text
38 passwordField = new JPasswordField( "Hidden text" );
39 add( passwordField ); // add passwordField to JFrame
40
41 // register event handlers
42
43 textField1.addActionListener( this );
44 textField2.addActionListener( this );
45 textField3.addActionListener( this );
46 passwordField.addActionListener( this );
47 } // end TextFieldFrame constructor
48
49
50
51
52 // process text field events
53 public void actionPerformed((ActionEvent event)
54 {
55     String string = ""; // declare string to display
56

```



```

57 // user pressed Enter in JTextField textField1
58 if ( event.getSource() == textField1 )
59     string = String.format( "textField1: %s",
60                             event.getActionCommand() );
61
62 // user pressed Enter in JTextField textField2
63 else if ( event.getSource() == textField2 )
64     string = String.format( "textField2: %s",
65                             event.getActionCommand() );
66
67 // user pressed Enter in JTextField textField3
68 else if ( event.getSource() == textField3 )
69     string = String.format( "textField3: %s",
70                             event.getActionCommand() );
71
72 // user pressed Enter in JTextField passwordField
73 else if ( event.getSource() == passwordField )
74     string = String.format( "passwordField: %s",
75                             new String( passwordField.getPassword() ) );
76
77 // display JTextField content
78 JOptionPane.showMessageDialog( null, string )
79 } // end method actionPerformed
80 } // end private inner class TextFieldHandler
81 } // end class TextFieldFrame

```

Проверява дали събитието е породено от textField1

Прочита текст въведен в тестовото поле

Проверява дали събитието е породено от textField2

Прочита текст въведен в тестовото

Проверява дали събитието е породено от textField3

Прочита текст въведен в тестовото поле

Проверява дали събитието е породено от passwordField

Прочита паролата въведена в passwordField

TextFieldFrame  
.java

(3 от 3)



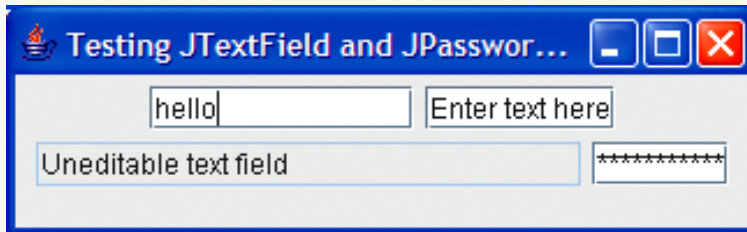
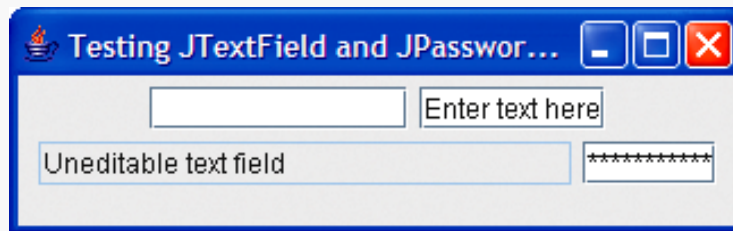
## Outline

### TextFieldTest .java

(1 от 2)

```

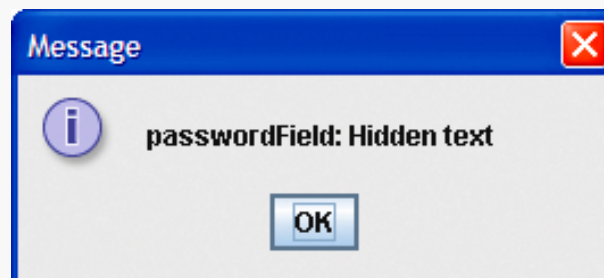
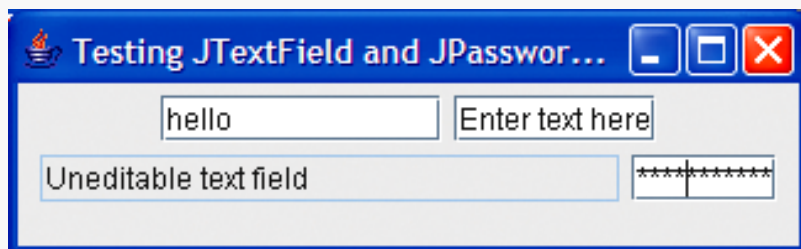
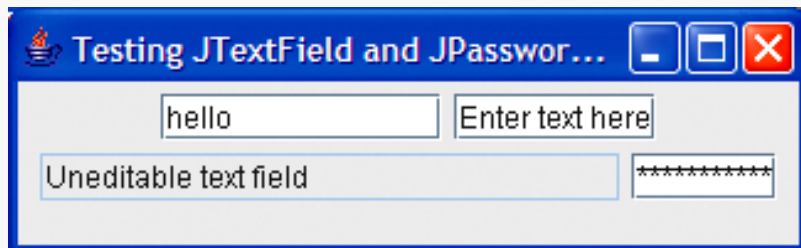
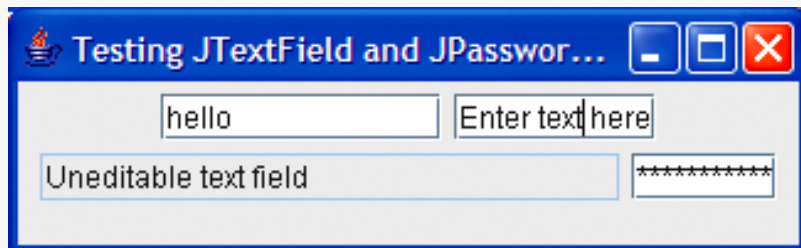
1 // Fig. 11.10: TextFieldTest.java
2 // Testing TextFieldFrame.
3 import javax.swing.JFrame;
4
5 public class TextFieldTest
6 {
7     public static void main( String args[] )
8     {
9         TextFieldFrame textFieldFrame = new TextFieldFrame();
10        textFieldFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        textFieldFrame.setSize( 325, 100 ); // set frame size
12        textFieldFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class TextFieldTest
  
```



## Outline

### TextFieldTest .java

(2 of 2)



## 11.5 Текстови полета и въведение към обработка на събитие `ActionEvent` с **вложени класове**

- Вложен (**вътрешен**) клас е клас, чиято **дефиниция се съдържа** изцяло в (вътре) дефиницията на **друг клас**
- При обработка на събития вътрешен клас обикновено се използва да **“пакетира”** метод(ите) за обработка на събитието
- За всяко събитие има определен *interface*, който определя **методите**, с които може да се обработва това събитие.
- Обект от **вътрешния клас**, който **реализира** дадения **интерфейс**, трябва да се **регистрира** към съответната компонента, за да може тази **компонента да реагира** на събитието, по начина по който са **реализирани методите на интерфейса**

## Outline

TextFieldFrame  
.java

(1 от 3)

```
1 // Fig. 11.9: TextFieldFrame.java
2 // Demonstrating the JTextField class.
3 import java.awt.FlowLayout;
4 import java.awt.event.ActionListener;
5 import java.awt.event.ActionEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JTextField;
8 import javax.swing.JPasswordField;
9 import javax.swing.JOptionPane;
10
11 public class TextFieldFrame extends JFrame
12 {
13     private JTextField textField1; // text field with set size
14     private JTextField textField2; // text field constructed with text
15     private JTextField textField3; // text field with text and size
16     private JPasswordField passwordField; // password field with text
17
18     // TextFieldFrame constructor adds JTextFields to JFrame
19     public TextFieldFrame()
20     {
21         super( "Testing JTextField and JPasswordField" );
22         setLayout( new FlowLayout() ); // set frame layout
23
24         // construct textfield with 10 columns
25         textField1 = new JTextField( 10 );
26         add( textField1 ); // add textField1 to JFrame
27     }
28 }
```

Външният клас **не**  
**имплементира**  
**ActionListener.**  
Събитието **ActionEvent**  
ще **се обработва във**  
**вътрешен клас**

Създава обект **JTextField**



## Outline

TextFieldFrame  
.java

(2 от 3)

Създава обект JTextField

Създава обект JTextField  
без да позволяваСъздава обект  
JPasswordFieldСъздава обект за обработка на  
ActionEvent event handlerРегистрира обект за обработка на  
ActionEvent към **всяка компонента**Създава **вътрешен клас**  
**реализиращ метод за обработка**  
на събитието определен от  
interface ActionListenerДефинира метод **actionPerformed** за обработка на  
събитието

```

28 // construct textfield with default text
29 textField2 = new JTextField( "Enter text here" );
30 add( textField2 ); // add textField2 to JFrame
31
32 // construct textfield with default text and 21 columns
33 textField3 = new JTextField( "Uneditable text field", 21 );
34 textField3.setEditable( false ); // disable editing
35 add( textField3 ); // add textField3 to JFrame
36
37 // construct passwordfield with default text
38 passwordField = new JPasswordField( "Hidden text" );
39 add( passwordField ); // add passwordField to JFrame
40
41 // register event handlers
42 TextFieldHandler handler = new TextFieldHandler();
43 textField1.addActionListener( handler );
44 textField2.addActionListener( handler );
45 textField3.addActionListener( handler );
46 passwordField.addActionListener( handler );
47 } // end TextFieldFrame constructor
48
49 // private inner class for event handling
50 private class TextFieldHandler implements ActionListener
51 {
52     // process text field events
53     public void actionPerformed((ActionEvent event)
54     {
55         String string = ""; // declare string to display
56

```



```

57 // user pressed Enter in JTextField textField1
58 if ( event.getSource() == textField1 )
59     string = String.format( "textField1: %s",
60                             event.getActionCommand() );
61
62 // user pressed Enter in JTextField textField2
63 else if ( event.getSource() == textField2 )
64     string = String.format( "textField2: %s",
65                             event.getActionCommand() );
66
67 // user pressed Enter in JTextField textField3
68 else if ( event.getSource() == textField3 )
69     string = String.format( "textField3: %s",
70                             event.getActionCommand() );
71
72 // user pressed Enter in JTextField passwordField
73 else if ( event.getSource() == passwordField )
74     string = String.format( "passwordField: %s",
75                             new String( passwordField.getPassword() ) );
76
77 // display JTextField content
78 JOptionPane.showMessageDialog( null, string )
79 } // end method actionPerformed
80 } // end private inner class TextFieldHandler
81 } // end class TextFieldFrame

```

Проверява дали събитието е породено от textField1

Прочита текст въведен в тестовото поле

Проверява дали събитието е породено от textField2

Прочита текст въведен в тестовото

Проверява дали събитието е породено от textField3

Прочита текст въведен в тестовото поле

Проверява дали събитието е породено от passwordField

Прочита паролата въведена в passwordField

TextFieldFrame  
.java

(3 от 3)



## Outline

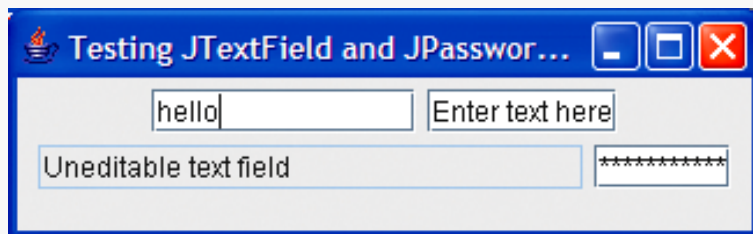
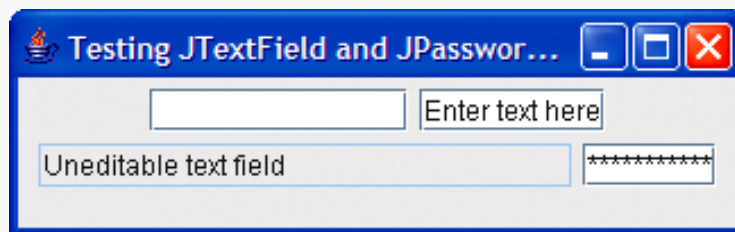
### TextFieldTest .java

(1 от 2)

```

1 // Fig. 11.10: TextFieldTest.java
2 // Testing TextFieldFrame.
3 import javax.swing.JFrame;
4
5 public class TextFieldTest
6 {
7     public static void main( String args[] )
8     {
9         TextFieldFrame textFieldFrame = new TextFieldFrame();
10        textFieldFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        textFieldFrame.setSize( 325, 100 ); // set frame size
12        textFieldFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class TextFieldTest

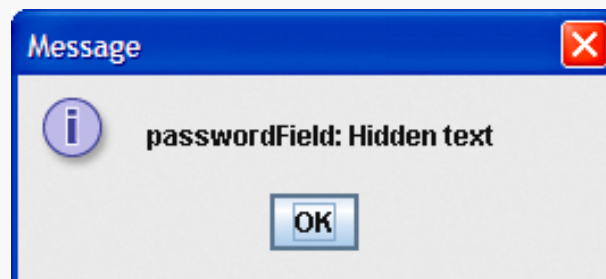
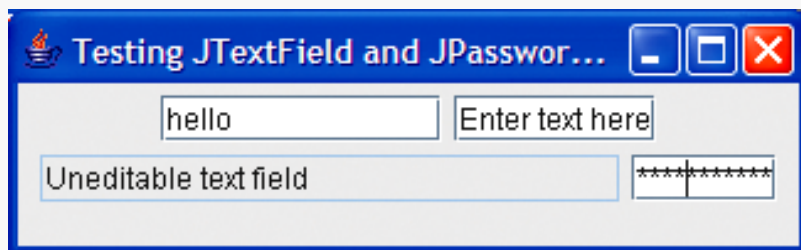
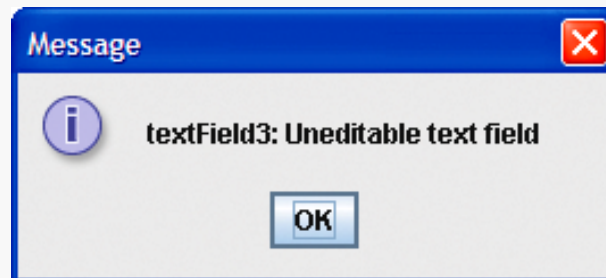
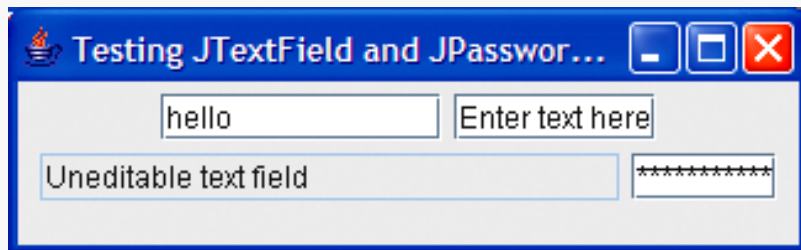
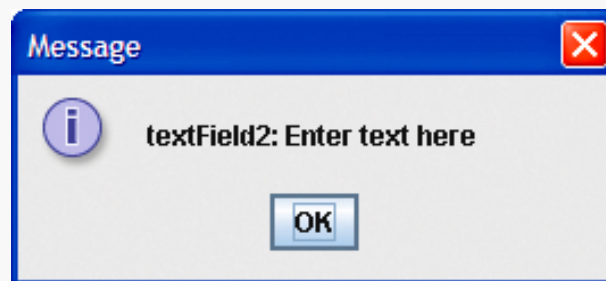
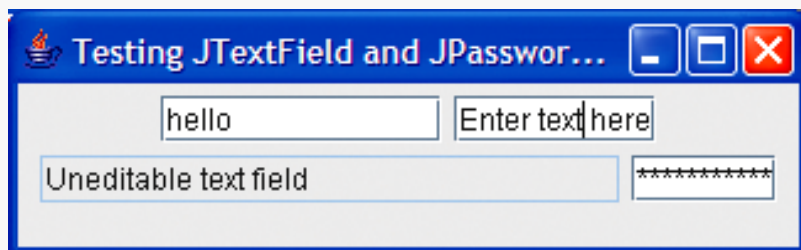
```



## Outline

### TextFieldTest .java

(2 of 2)



# Стъпки, необходими за обработване на събития от графичния интерфейс

Следните стъпки са необходими:

1. Създаване на клас за “*пакетиране*” на метод за обработка на събитието- той имплементира стандартен (библиотечен) интерфейс съответстващ по име на събитието  
(събитие *ActionEvent*  $\leftrightarrow$  интерфейс *ActionListener*)
2. Реализиране на методите на интерфейса в този клас съобразно събитието, което ще се обработва  
(*например* `public void actionPerformed()`)
3. Регистриране на обект (*например*, *actionHandler*) от класа с реализацията метода(ите) за обработка на събитието към графичната компонента (*например*, *componentRefVar*), която ще реагира на (*слуша за*) събитието (*например*, *ActionEvent*)

(*например*  
`componentRefVar.addActionListener(actionHandler)` )

# Използване на вложен клас за реализиране на обработката на събитието **накратко**

- Класове на горно ниво
  - Не са вложени в други класове
- Вложени класове
  - Декларират се вътре в друг клас
  - не-СТАТИЧНО вложените класове се наричат вложени “вътрешни” (*inner*) класове
  - Най- често се използват при обработка на събития
  - Всеки вложен клас има директен достъп до членовете на обхващащия го клас, дори и когато тези членове са `private`.

# Използвана на вложен клас за обработка на събитие

- *JTextField* и *JPasswordField* КОМПОНЕНТИ
  - Натискане на бутон на мишката във всяко от тези компоненти води до пораждање на събитието *ActionEvent*
    - Обработката на това събитие се извършва, чрез класове имплементиращи *ActionListener* interface
    - Този интерфейс изисква реализирането на метод *actionPerformed()*, който служи за обработка на този тип събитие

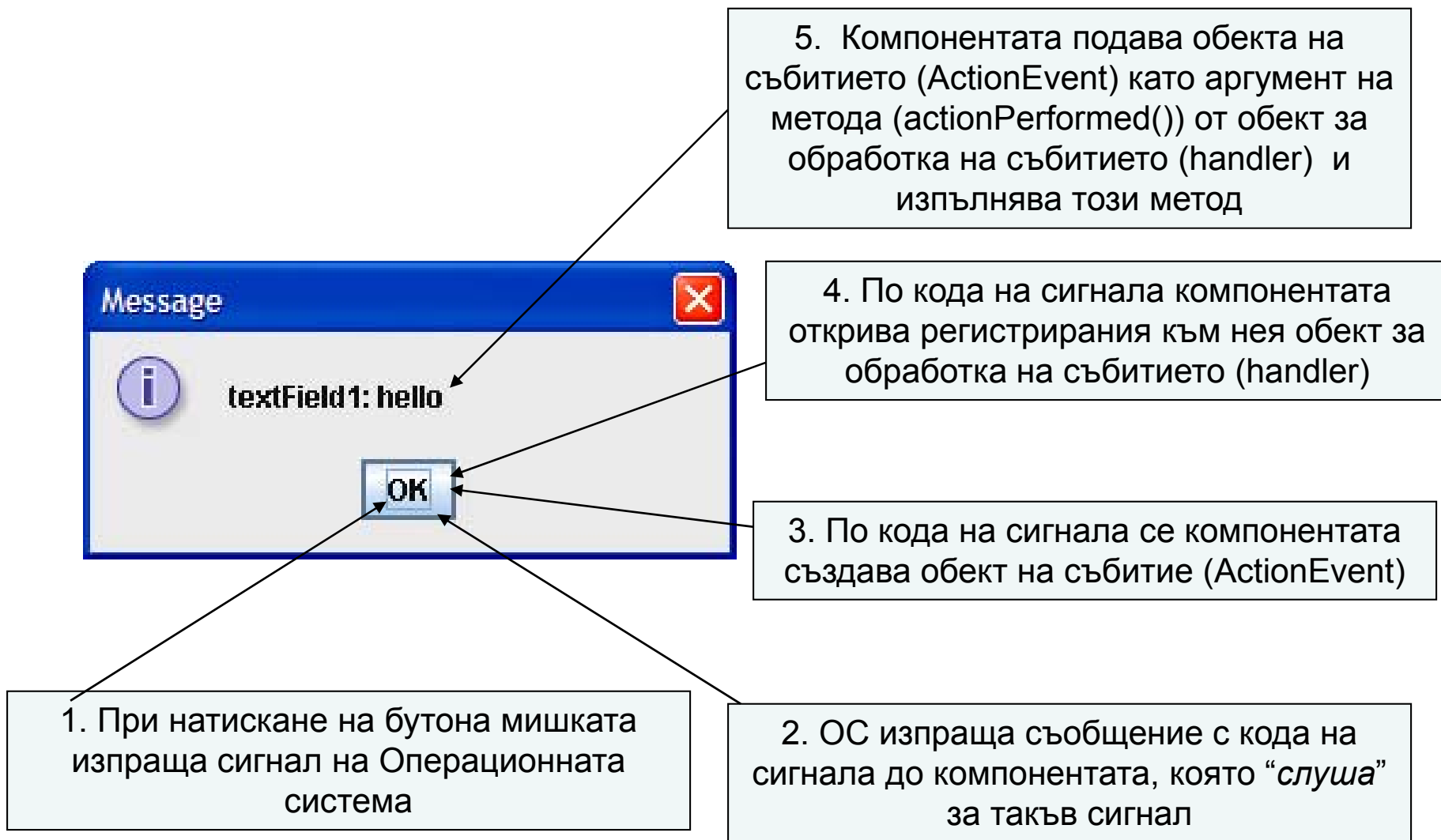
# Регистриране на обработчика на събитието

- Регистрирането се извършва като се
  - Извиква метод `addActionListener` за регистриране на обект от типа на интерфейса `ChangeListener` (преобразуване нагоре до типа на интерфейса!)
  - `ChangeListener` обектът започва да “слуша” за събития от типа на ***ActionEvent***
  - *Пропускането да се регистрира обработчика на събитието, не позволява на приложението да реагира на това събитие*

# Използване на метода actionPerformed

- Взима за аргумент обект от породеното събитие (**ActionEvent**)
  - този обект се предава на actionPerformed от източника на събитието (Event source)
  - източникът на събитието е компонентата, от която е породила събитието (създава е обектът от клас **ActionEvent**)
  - Референция към източника на събитието се съхранява в обекта на събитието и може да се получи с метода getSource
  - Когато източникът е текстово поле JTextField, текстът в него се получава с getActionCommand или getText
  - Текст от JPasswordField се получава с using getPassword

# Обработване на събитие

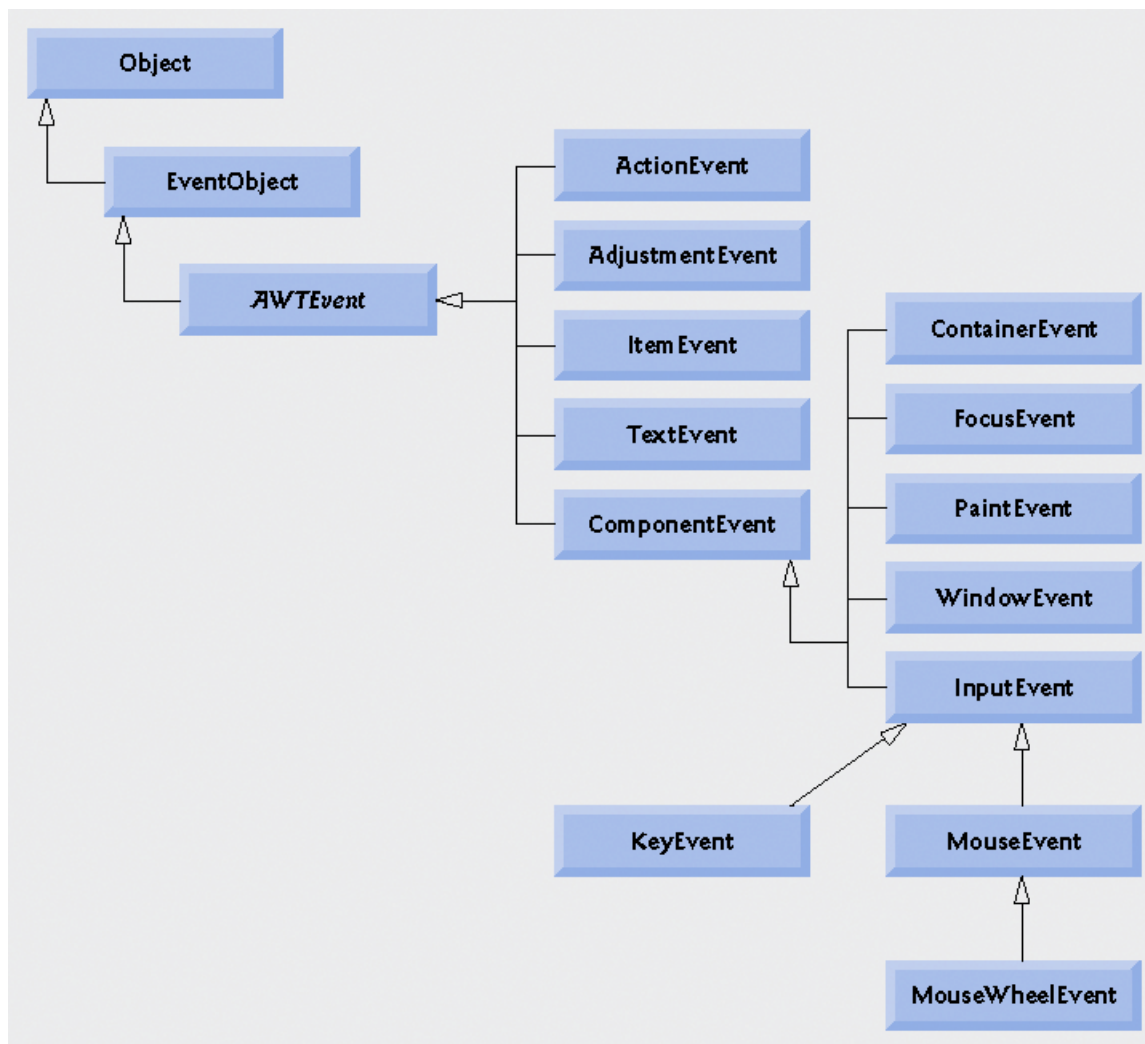


## 11.6 Общи типове събития и съответните им интерфейс-и

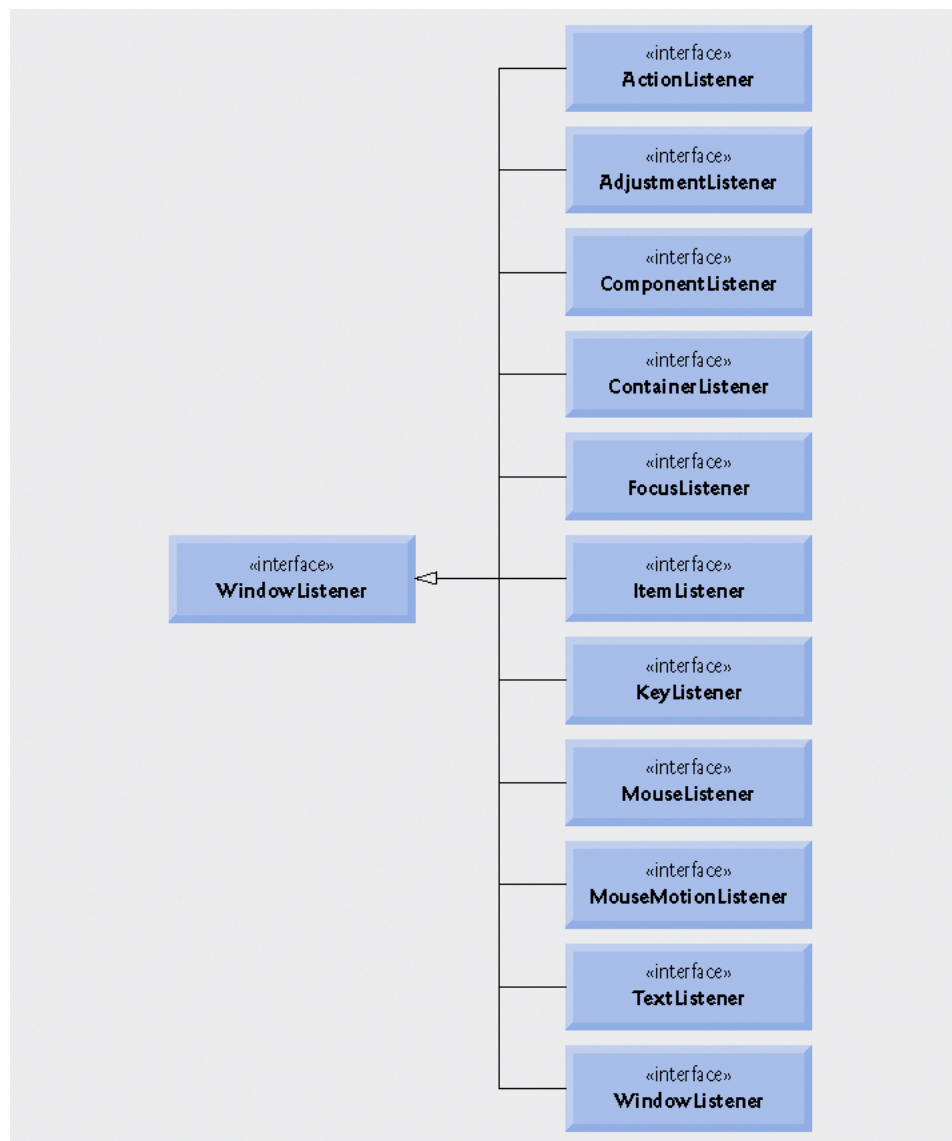
- Типове събития
  - Производни на клас `AWTEvent`
  - Повечето са декларирани в `package java.awt.event`
  - Други, специфични за `Swing` компоненти са декларирани в `javax.swing.event`

## 11.6 Общи типове събития и съответните им интерфейс-и

- **Модел на делегиране при обработка на събитията**
  - **Източникът на събитието е тази компонента с която потребителят взаимодейства**
  - **Обектът на събитието се създава и съдържа информация за събитието, което е породено**
  - **Слушател (listener) на събитието получава съобщение за станалото събитие т.е. изпълнява съответен метод за обработка на събитието**



**Fig. 11.11** | Някои от основните събития в package `java.awt.event`.



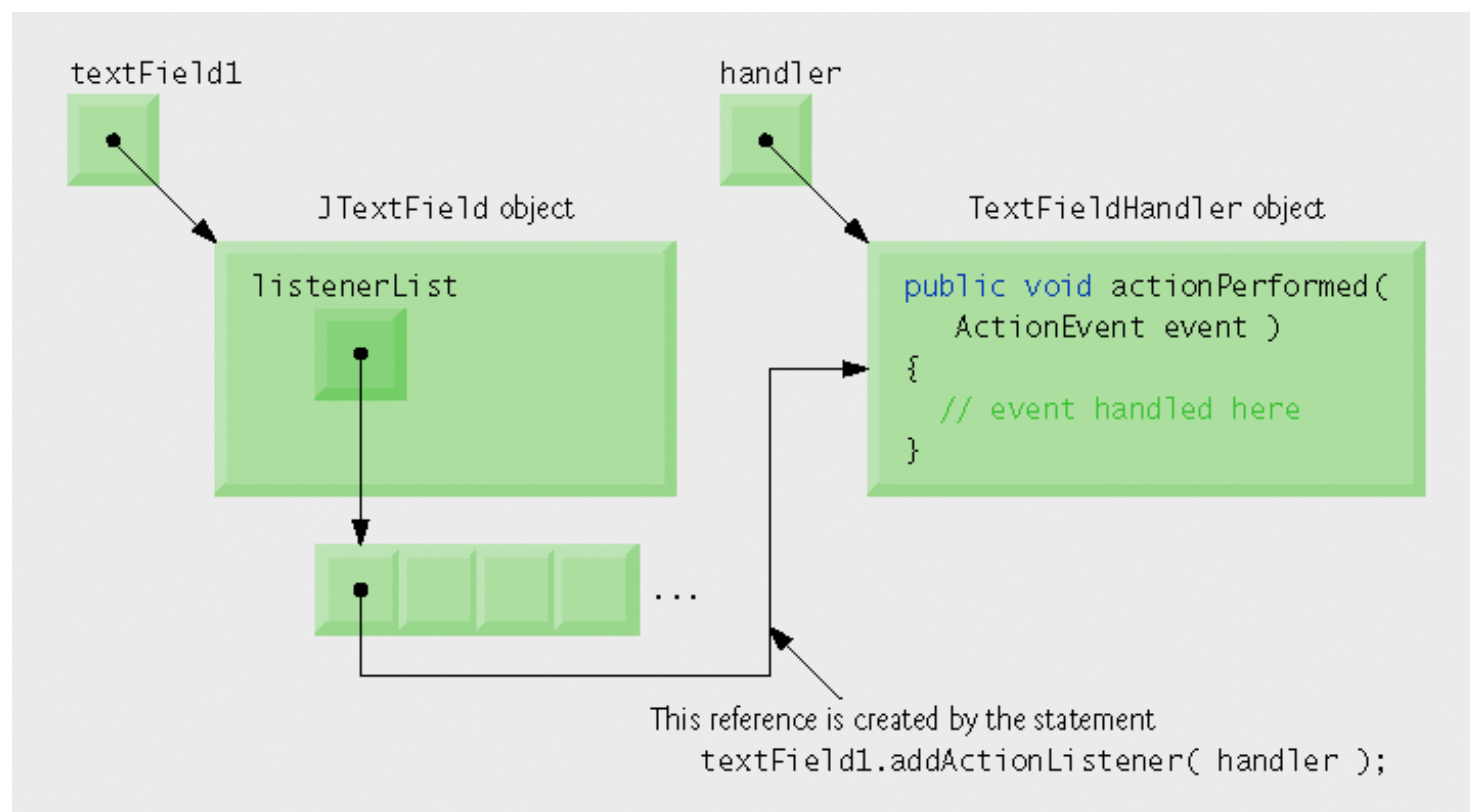
**Fig. 11.12** | Някой от основните събития в `package java.awt.event`.

## 11.7 Схематично представяне на модела за обработка на събития в Java

- **Оставащи въпроси за изясняване**
  - **Как се регистрира обектът от класа за обработка на събитието?**
  - **Как графичната компонента знае да изпълни `actionPerformed` а не някой друг метод?**

# Регистриране на събитията

- Всеки обект от клас произведен на *JComponent* има данна *listenerList* от *class EventListenerList*, която има списък с референции към всички слушатели (обработчици на събития) регистрирани с тази компонента
- Този списък е в съответствие с уникалния код на сигнала подаван от ОС при възникване на събитието
  - Например, при изпълнение на  
*textField1.addActionListener( handler );*
- се добавя нова референция към данната *listenerList* на обекта *textField1*
- обектът *textField1* ще приема сигнали от ОС (чрез JVM) при възникване на *ActionEvent*



**Fig. 11.13 | Регистриране на обект за обработка на събитие към `JTextField` `textField1`.**

# Изпълнение на метода за обработка

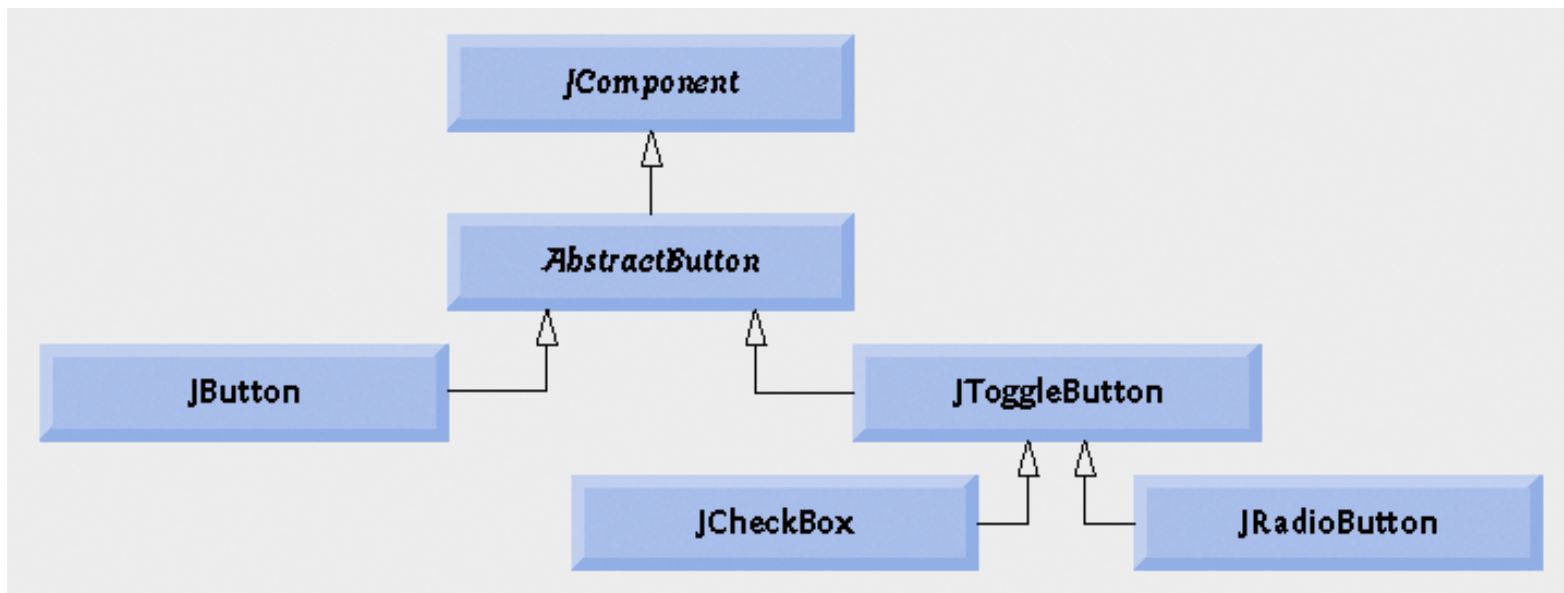
- Събитията се прехвърлят за обработка на обекти от съответния им тип
  - Всяко събитие прихванато от ОС има уникален идентификатор за разпознаване на типа му
  - Графичната компонента получава този идентификатор от ОС и го използва да открие регистриран към нея обект за обработка на това събитие (проверява се дали на съответното място в списъка *listenerList* има *референция към обект*)
  - Ако се открие регистриран обект за обработка на събитието, то се изпълнява метод в този обект, съобразно имплементирания в него интерфейс
- MouseEvent се обработват от MouseListener и MouseMotionListener обекти
- KeyEvent се обработват от KeyListener обект и пр.

# 11.8 JButton

- Бутон
    - Компонентата “*слуша*” за **натискане на бутон** на мишката или **Return**, когато компонентата е на фокус
    - Реализира се като
      - обикновен button,
      - check box,
      - toggle button или
      - radio button
    - Всичките ѝ реализации са класове производни на class `AbstractButton`
- Следствие: Понеже class `AbstractButton` дефинира методи за извеждане на текст и картинки, то всички производни класове на `AbstractButton` също позволяват извеждане на текст и картинки.

## 11.8 JButton

- **Обикновен (команден) бутон**
  - Поражда събитие `ActionEvent` при натискане на бутон на мишка или `Return` когато има фокус на приложението
  - Обект на `class JButton`
  - Изобразява текст с информация за действието, което изпълнява (връща се с метод `getActionCommand()` на `ActionEvent` обект)



**Fig. 11.14 | Swing йерархия от бутони.**

## Outline

ButtonFrame.java

(1 от 2)

1 // Fig. 11.15: ButtonFrame.java

2 // Creating JButtons.

3 import java.awt.FlowLayout;

4 import java.awt.event.ActionListener;

5 import java.awt.event.ActionEvent;

6 import javax.swing.JFrame;

7 import javax.swing.JButton;

8 import javax.swing.Icon;

9 import javax.swing.ImageIcon;

10 import javax.swing.JOptionPane;

11

12 public class ButtonFrame extends JFrame

13 {

14 private JButton plainJButton; // button with just text

15 private JButton fancyJButton; // button with icons

16

17 // ButtonFrame adds JButtons to JFrame

18 public ButtonFrame()

19 {

20 super( "Testing Buttons" );

21 setLayout( new FlowLayout() ); // set frame layout

22

23 plainJButton = new JButton( "Plain Button" ); // button with text

24 add( plainJButton ); // add plainJButton to JFrame

25

26 Icon bug1 = new ImageIcon( getClass().getResource( "bug1.gif" ) );

27 Icon bug2 = new ImageIcon( getClass().getResource( "bug2.gif" ) );

28 fancyJButton = new JButton( "Fancy Button", bug1 ); // set image

29 fancyJButton.setRolloverIcon( bug2 ); // set rollover image

30 add( fancyJButton ); // add fancyJButton to JFrame

Декларира две променливи от клас JButton

Създава нов обект JButton

Създава две иконки ImageIcon

Създава нов JButton

Задава икона за JButton при попадане на курсор на мишката върху бутона



Е. Кръстев, Увод в  
Програмирането  
2007

## Outline

ButtonFrame.java

(2 от 2)

```

31 // create new ButtonHandler for button event handling
32 ButtonHandler handler = new ButtonHandler();
33 fancyJButton.addActionListener( handler );
34 plainJButton.addActionListener( handler );
35 } // end ButtonFrame constructor
36
37 // inner class for button event handling
38 private class ButtonHandler implements ActionListener
39 {
40     // handle button event
41     public void actionPerformed((ActionEvent event)
42     {
43         JOptionPane.showMessageDialog( ButtonFrame.this, String.format(
44             "You pressed: %s", event.getActionCommand() ) );
45     } // end method actionPerformed
46 } // end private inner class ButtonHandler
47 } // end class ButtonFrame
48

```

Създава слушател на събитие

Регистрира слушател на събитие

Вътрешният клас  
имплементира интерфейс  
ActionListener

Извежда текста написан върху  
натиснатия JButton

Достъп до текущия обект от външния клас –  
позволява да се центрира диалоговия прозорец  
спрямо графичния прозорец

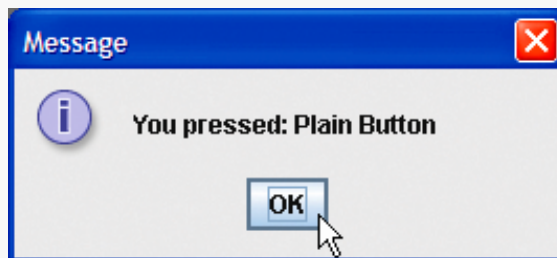
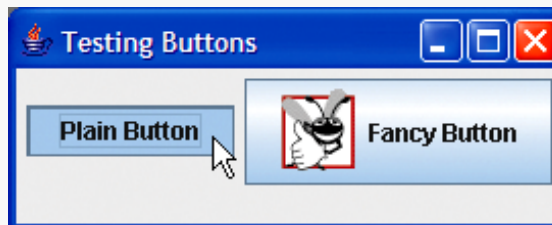
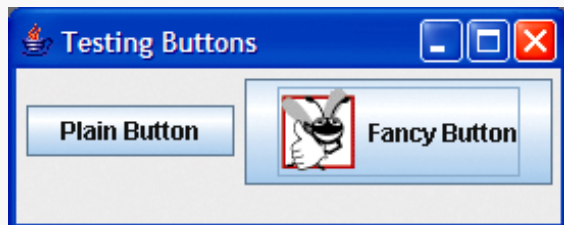


## Outline

### ButtonTest.java

(1 of 2)

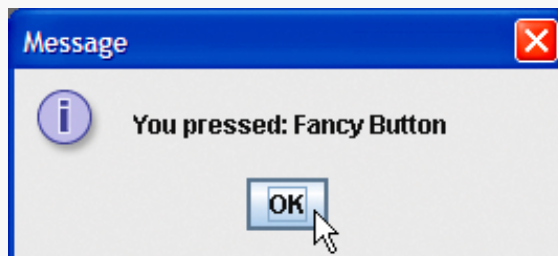
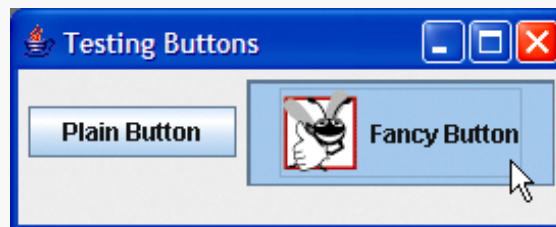
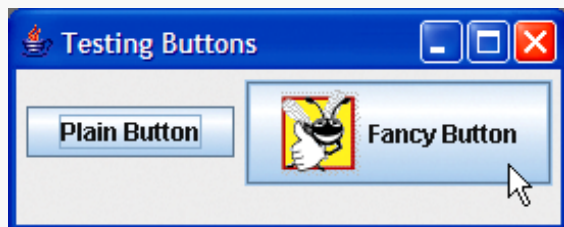
```
1 // Fig. 11.16: ButtonTest.java
2 // Testing ButtonFrame.
3 import javax.swing.JFrame;
4
5 public class ButtonTest
6 {
7     public static void main( String args[] )
8     {
9         ButtonFrame buttonFrame = new ButtonFrame(); // create ButtonFrame
10        buttonFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        buttonFrame.setSize( 275, 110 ); // set frame size
12        buttonFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class ButtonTest
```



## Outline

### ButtonTest.java

(2 от 2)



## 11.8 JButton

- **JButtons може да има и rollover icon**
  - **rollover icon се появява при преминаване на мишката върху бутона**
  - **Добавя се към JButton с метода setRolloverIcon**

# Software Engineering факт 11.4

---

При използване на `this` референцията във вътрешен клас се реферира текущия обект от вътрешния клас.

Даден метод от вътрешния може да реферира текущия обект от външния клас като използва името на външния клас следвано от точка и референцията `this` (*както в примера програма `ButtonFrame.this`*)

## 11.13 Обработка на събития породени от мишката

- Видове методи
  - a) Породени при **статично положение** на мишката  
`interface MouseListener`
  - b) Породени при **движение** на мишката  
`interface MouseMotionListener`
    - a) и b) Използват за аргумент `MouseEvent`
  - Породени при **движение** на колелото на мишката  
`interface MouseWheelListener`  
Използват за аргумент `MouseWheelEvent`

## Методи на интерфейсите **MouseListener** и **MouseMotionListener**

### *Методи на interface **MouseListener***

```
public void mousePressed( MouseEvent event )
```

Изпълнява се при натискане на бутон върху компонентата.

```
public void mouseClicked( MouseEvent event )
```

Изпълнява се при натискане и освобождаване на бутон върху компонентата като мишката остава неподвижна. Преди този метод се изпълнява **mousePressed**.

```
public void mouseReleased( MouseEvent event )
```

Изпълнява се при освобождаване на бутон след като е бил натиснат върху компонентата след движение на мишката. . Преди този метод се изпълнява **mousePressed** и един или няколко **mouseDragged**.

```
public void mouseEntered( MouseEvent event )
```

Изпълнява се при навлизане на мишката в очертанията на компонентата.

**Fig. 11.27 | Методи на интерфейсите **MouseListener** и **MouseMotionListener**. (1 от 2.)**

## Методи на интерфейсите **MouseListener** и **MouseMotionListener**

```
public void mouseExited( MouseEvent event )
```

Изпълнява се при излизане на мишката извън компонентата.

*Методи на interface MouseMotionListener*

```
public void mouseDragged( MouseEvent event )
```

Изпълнява се при движение на мишката върху компонентата с постоянно натиснат бутон (drag). Преди първото изпълнение се извиква `mousePressed`.

```
public void mouseMoved( MouseEvent event )
```

Изпълнява се при движение на мишката върху компонентата без натиснат бутон.

**Fig. 11.27 | Методи на интерфейсите `MouseListener` и `MouseMotionListener`. (2 от 2.)**

## Outline

### MouseTracker Frame.java

(1 от 4)

```

1 // Fig. 11.28: MouseTrackerFrame.java
2 // Demonstrating mouse events.
3 import java.awt.Color;
4 import java.awt.BorderLayout;
5 import java.awt.event.MouseListener;
6 import java.awt.event.MouseMotionListener;
7 import java.awt.event.MouseEvent;
8 import javax.swing.JFrame;
9 import javax.swing.JLabel;
10 import javax.swing.JPanel;
11
12 public class MouseTrackerFrame extends JFrame
13 {
14     private JPanel mousePanel; // panel in which mouse events will occur
15     private JLabel statusBar; // label that displays event information
16
17     // MouseTrackerFrame constructor sets up GUI and
18     // registers mouse event handlers
19     public MouseTrackerFrame()
20     {
21         super( "Demonstrating Mouse Events" );
22
23         mousePanel = new JPanel(); // create panel
24         mousePanel.setBackground( Color.WHITE ); // set background color
25         add( mousePanel, BorderLayout.CENTER ); // add panel to JFrame
26
27         statusBar = new JLabel( "Mouse outside JPanel" );
28         add( statusBar, BorderLayout.SOUTH ); // add label to JFrame
29

```

Създаваме `JPanel` за локализиране на събитията на мишката в него- ще рисуваме в него

Задаваме white фон (background)

Създаваме `JLabel` и го добавяме към приложението



```

30 // create and register listener for mouse and mouse motion events
31 MouseHandler handler = new MouseHandler();
32 mousePanel.addMouseListener( handler );
33 mousePanel.addMouseMotionListener( handler );
34 } // end MouseTrackerFrame constructor
35
36 private class MouseHandler implements MouseListener,
37     MouseMotionListener
38 {
39     // MouseListener event handlers
40     // handle event when mouse released immediately after press
41     public void mouseClicked( MouseEvent event )
42     {
43         statusBar.setText( String.format( "Clicked at [%d, %d]",
44             event.getX(), event.getY() ) );
45     } // end method mouseClicked
46
47     // handle event when mouse pressed
48     public void mousePressed( MouseEvent event )
49     {
50         statusBar.setText( String.format( "Pressed at [%d, %d]",
51             event.getX(), event.getY() ) );
52     } // end method mousePressed
53
54     // handle event when mouse released after dragging
55     public void mouseReleased( MouseEvent event )
56     {
57         statusBar.setText( String.format( "Released at [%d, %d]",
58             event.getX(), event.getY() ) );
59     } // end method mouseReleased

```

Създаваме event handler обект за  
обработване на събития от мишката

Регистрираме обекта за обработка на  
събитията

Имплементираме интерфейсите на  
събитията

Декларираме метод mouseClicked

Извеждаме “клик” координатите

Декларираме метод mousePressed

Декларираме метод mouseReleased

MouseTracker  
Frame.java

(2 от 4)



```
60 // handle event when mouse enters area
61 public void mouseEntered( MouseEvent event
62 {
63     statusBar.setText( String.format( "Mouse entered at [%d, %d]",
64         event.getX(), event.getY() ) );
65     mousePanel.setBackground( Color.GREEN );
66 } // end method mouseEntered
67
68 // handle event when mouse exits area
69 public void mouseExited( MouseEvent event )
70 {
71     statusBar.setText( "Mouse outside JPanel" );
72     mousePanel.setBackground( Color.WHITE );
73 } // end method mouseExited
74
75
```

Декларираме метод mouseEntered method

Задаваме green фон  
(background) на JPanel

Декларираме метод  
mouseExited

Задаваме white фон  
(background) на JPanel

## Outline

MouseListener  
Frame.java

(3 от 4)



## Outline

```
76 // MouseMotionListener event handlers
77 // handle event when user drags mouse with button pressed
78 public void mouseDragged( MouseEvent event )
79 {
80     statusBar.setText( String.format( "Dragged at [%d, %d]",
81         event.getX(), event.getY() ) );
82 } // end method mouseDragged
83
84 // handle event when user moves mouse
85 public void mouseMoved( MouseEvent event )
86 {
87     statusBar.setText( String.format( "Moved at [%d, %d]",
88         event.getX(), event.getY() ) );
89 } // end method mouseMoved
90 } // end inner class MouseHandler
91 } // end class MouseTrackerFrame
```

Декларираме метод mouseDragged

Декларираме метод mouseMoved

MouseTracker  
Frame.java

(4 от 4)

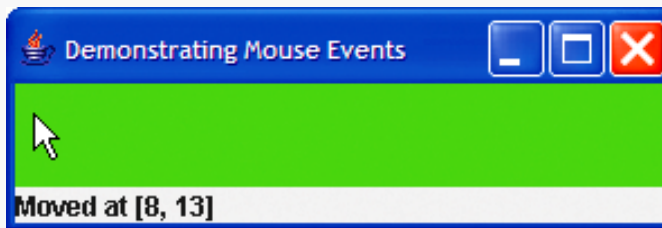
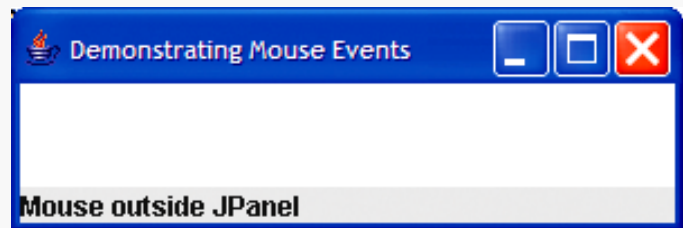


## Outline

### MouseListener Frame.java

(1 of 2)

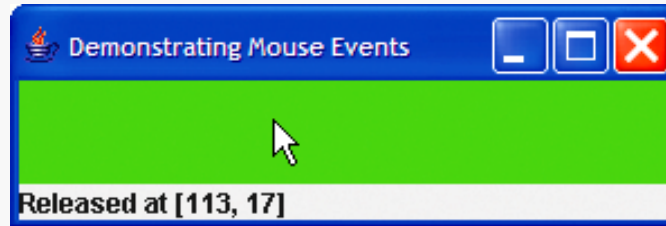
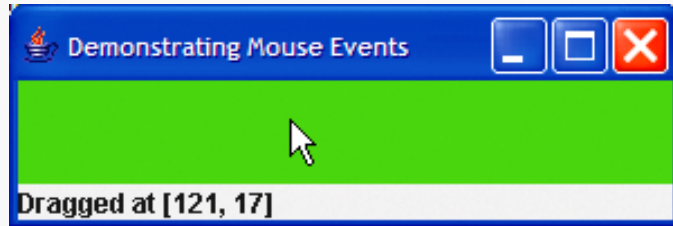
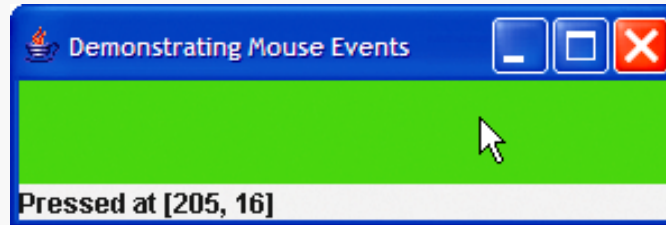
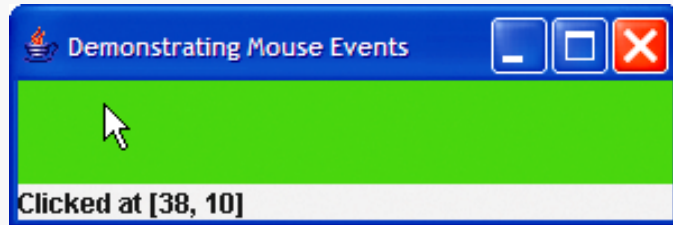
```
1 // Fig. 11.29: MouseTrackerFrame.java
2 // Testing MouseTrackerFrame.
3 import javax.swing.JFrame;
4
5 public class MouseTracker
6 {
7     public static void main( String args[] )
8     {
9         MouseTrackerFrame mouseTrackerFrame = new MouseTrackerFrame();
10        mouseTrackerFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        mouseTrackerFrame.setSize( 300, 100 ); // set frame size
12        mouseTrackerFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class MouseTracker
```



## Outline

MouseListener  
Frame.java

(2 of 2)



## 11.14 Adapter класове

- **Позволява да се предефинират само определени методи от интерфейс на дадено събитие**
- **Това е клас, който**
  - **Имплементира интерфейс на събитие**
  - **Всеки от методите на интерфейса е дефиниран по подразбиране (празен блок команди)**

# Пример: MouseAdapter

- MouseAdapter
  - Реализира методите на интерфейс `MouseListener`
- MouseMotionAdapter
  - Реализира методите на интерфейс `MouseMotionListener`
  - Позволява да се предефинират само тези от методите, които са необходими за конкретното приложение

Event-adapter class in java.awt.event	Implements interface
<b>Component</b> Adapter	<b>Component</b> Listener
<b>Container</b> Adapter	<b>Container</b> Listener
<b>Focus</b> Adapter	<b>Focus</b> Listener
<b>Key</b> Adapter	<b>Key</b> Listener
<b>Mouse</b> Adapter	<b>Mouse</b> Listener
<b>MouseMotion</b> Adapter	<b>MouseMotion</b> Listener
<b>Window</b> Adapter	<b>Window</b> Listener

**Fig. 11.30 |** Адаптер класове и интерфейсите които имплементират  
в  
package java.awt.event.

## Outline

### MouseDetails Frame.java

(1 от 2)

```
1 // Fig. 11.31: MouseDetailsFrame.java
2 // Demonstrating mouse clicks and distinguishing between mouse buttons.
3 import java.awt.BorderLayout;
4 import java.awt.Graphics;
5 import java.awt.event.MouseAdapter;
6 import java.awt.event.MouseEvent;
7 import javax.swing.JFrame;
8 import javax.swing.JLabel;
9
10 public class MouseDetailsFrame extends JFrame
11 {
12     private String details; // String representing
13     private JLabel statusBar; // JLabel that appears at bottom of window
14
15     // constructor sets title bar String and register mouse listener
16     public MouseDetailsFrame()
17     {
18         super( "Mouse clicks and buttons" );
19
20         statusBar = new JLabel( "Click the mouse" );
21         add( statusBar, BorderLayout.SOUTH );
22         addMouseListener( new MouseClickHandler() ); // add handler
23     } // end MouseDetailsFrame constructor
24
```

Регистрира обект за обработка на събитие



## Outline

MouseDetails  
Frame.java

(2 от 2)

```
25 // inner class to handle mouse events
26 private class MouseClickHandler extends MouseAdapter
27 {
28     // handle mouse click event and determine which button was pressed
29     public void mouseClicked( MouseEvent event )
30     {
31         int xPos = event.getX(); // get x position of mouse
32         int yPos = event.getY(); // get y position of mouse
33
34         details = String.format( "Clicked %d time(s)",
35             event.getClickCount() );
36
37         if ( event.isMetaDown() ) // right mouse button
38             details += " with right mouse button";
39         else if ( event.isAltDown() ) // middle mouse button
40             details += " with center mouse button";
41         else // left mouse button
42             details += " with left mouse button";
43
44         statusBar.setText( details ); // display message in statusBar
45     } // end method mouseClicked
46 } // end private inner class MouseClickHandler
47 } // end class MouseDetailsFrame
```

Брой натисканията на бутона

Проверява за натиснат десен  
бутон

Проверява за натиснат среден  
бутон

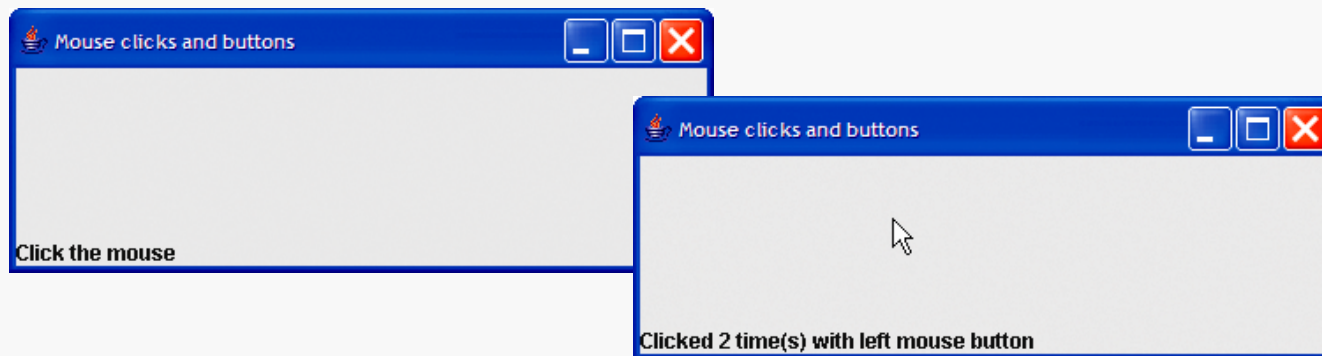


## Outline

### MouseDetails .java

(1 от 2)

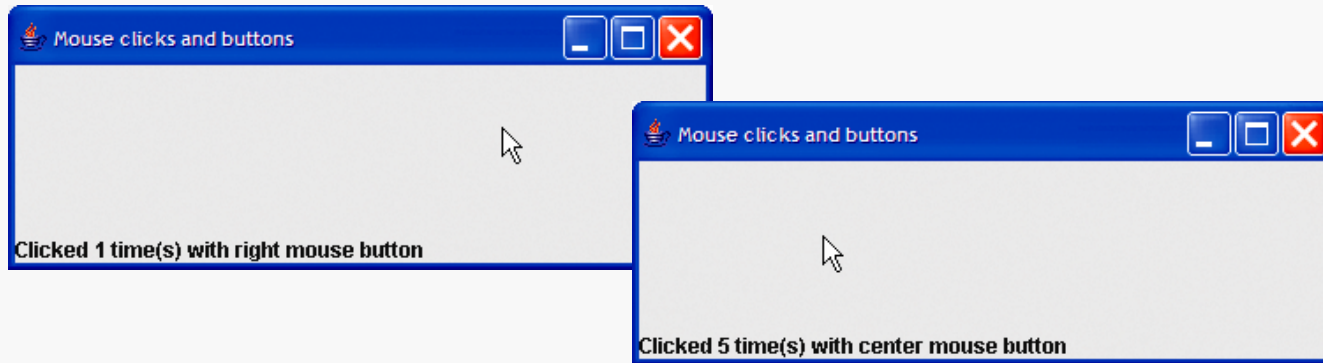
```
1 // Fig. 11.32: MouseDetails.java
2 // Testing MouseDetailsFrame.
3 import javax.swing.JFrame;
4
5 public class MouseDetails
6 {
7     public static void main( String args[] )
8     {
9         MouseDetailsFrame mouseDetailsFrame = new MouseDetailsFrame();
10        mouseDetailsFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        mouseDetailsFrame.setSize( 400, 150 ); // set frame size
12        mouseDetailsFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class MouseDetails
```



# Outline

## MouseDetails .java

(2 of 2)



Методи на MouseEvent	Описание
<code>isMetaDown()</code>	Връща <code>true</code> при натискане на десен бутон. .
<code>isAltDown()</code>	Връща <code>true</code> при натискане на среден бутон. .

**Fig. 11.33** | `InputEvent` методи, които позволяват да различим натискане на ляв, среден и десен бутон.

# 11.15 Пример: Използване на JPanel

## производен клас за рисуване с мишка

- **Предефинира class JPanel**
  - **Обособяваме област в графичния прозорец за рисуване**
  - **Може да използваме така дефинираната компонента многократно в различна комбинация и подреждане с други графични компоненти**

# Метод `paintComponent`

- Методът `paintComponent`
  - Служи за рисуване върху `Swing` компонента
  - Предефинирането му води до получаване на потребителски зададена рисунка
  - Трябва да извика метода `paintComponent` на базовия клас като първа изпълнима команда (прерисува графичната област заделена за компонентата)

# Технически данни 11.13

---

**Повечето Swing GUI компоненти могат да са прозрачни или плътни (непрозрачни). Когато Swing GUI компонента е зададена като непрозрачна, то нейният фон трябва да се изтрие при извикване на `paintComponent`. Само непрозрачни компоненти могат да изобразяват потребителски зададен фон. `JPanel` обектите са непрозрачни по подразбиране.**

# Дефиниране на потребителска област за рисуване

- Създаване на производен клас от `JPanel`
  - Обособява област за рисуване
  - клас `Graphics` се използва при рисуване върху Swing компоненти
  - клас `Point` от package `java.awt` представя точка с двойка *x-y* координати

## Outline

### PaintPanel.java

(1 от 2)

```
1 // Fig. 11.34: PaintPanel.java
2 // Using class MouseMotionAdapter.
3 import java.awt.Point;
4 import java.awt.Graphics;
5 import java.awt.event.MouseEvent;
6 import java.awt.event.MouseMotionAdapter;
7 import javax.swing.JPanel;
8
9 public class PaintPanel extends JPanel
10 {
11     private int pointCount = 0; // count number of points
12
13     // array of 10000 java.awt.Point references
14     private Point points[] = new Point[ 10000 ];
15
16     // set up GUI and register mouse event handler
17     public PaintPanel()
18     {
19         // handle frame mouse motion event
20         MouseMotionListener handler = new MouseMotionHandler();
21         addMouseMotionListener(handler);
22     } // end PaintPanel constructor
```

Създаваме масив от точки  
**Points**

Създаваме обект за обработка на  
събитието

Регистрираме обекта за  
обработка на събитието



```

22 private class MouseMotionHandler extends MouseMotionAdapter
23 {
24     // store drag coordinates and repaint
25     public void mouseDragged( MouseEvent event )
26     {
27         if ( pointCount < points.length )
28         {
29             points[ pointCount ] = event.getPoint(); // find point
30             pointCount++; // increment number of points in array
31             repaint(); // repaint JFrame
32         } // end if
33     } // end method mouseDragged
34 } // end MouseMotionHandler inner class
35
36
37
38 // draw oval in a 4-by-4 bounding box at specified location on window
39 public void paintComponent( Graphics g )
40 {
41     super.paintComponent( g ); // clears drawing area
42
43     // draw all points in array
44     for ( int i = 0; i < pointCount; i++ )
45         g.filloval( points[ i ].x, points[ i ].y, 4, 4 );
46 } // end method paintComponent
47 } // end class PaintPanel

```

Inner class for event handling

Предефинира метод mouseDragged method

Взема положението на курсора като обект от class Point

Перерисуване на JFrame

Взема x и y- координатите на Point елемент от масива

## Outline

PaintPanel.java

(2 от 2)



# Технически данни 11.14

---

**Извикването на `repaint` за дадена Swing GUI компонента води до прерисуването на компонентата при първи възможен момент. Фонът на тази GUI компонента се изтрива само ако компонентата е непрозрачна. Методът `setOpaque` на клас `JComponent` позволява с `boolean` аргумент да се укаже дали компонентата е непрозрачна (`true`) или прозрачна (`false`).**

# Технически данни 11.15

---

**При рисуване на върху GUI компонента координатното начало  $(0, 0)$  винаги е в горния ляв ъгъл на GUI тази компонента.**

## Outline

### Painter.java

(1 от 2)

```
1 // Fig. 11.35: Painter.java
2 // Testing PaintPanel.
3 import java.awt.BorderLayout;
4 import javax.swing.JFrame;
5 import javax.swing.JLabel;
6
7 public class Painter
8 {
9     public static void main( String args[] )
10    {
11        // create JFrame
12        JFrame application = new JFrame( "A simple paint program" );
13
14        PaintPanel paintPanel = new PaintPanel(); // create paint panel
15        application.add( paintPanel, BorderLayout.CENTER ); // in center
16
17        // create a label and place it in SOUTH of BorderLayout
18        application.add( new JLabel( "Drag the mouse to draw" ),
19                        BorderLayout.SOUTH );
20
21        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
22        application.setSize( 400, 200 ); // set frame size
23        application.setVisible( true ); // display frame
24    } // end main
25 } // end class Painter
```

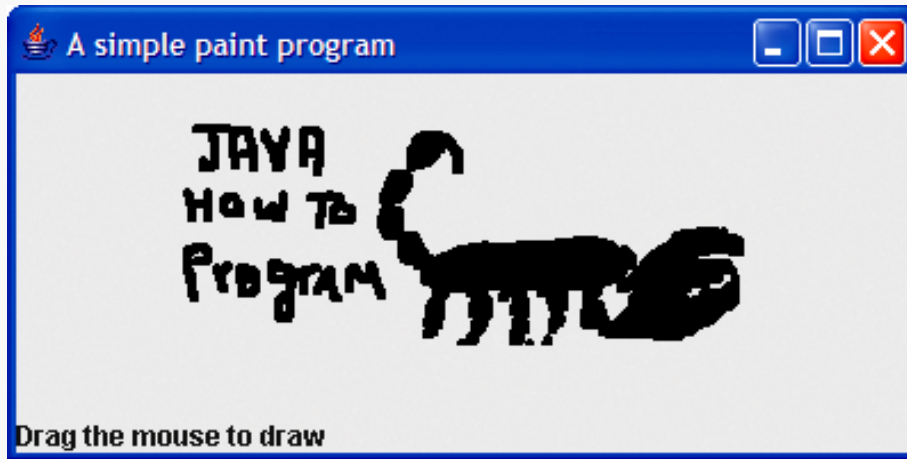
1. Създаваме обект от панела за рисуване
2. Добавяме го към графичния прозорец
3. Дефинираме параметрите на графичния прозорец
4. Изобразяваме прозореца с панела



## Outline

Painter.java

(2 от 2)



# 11.16 Обработка на събития от клавиатурата

- **interface KeyListener**
  - Обработва KeyEvent събития
  - Декларира методи
    - keyPressed – натискане на произволен клавиш
    - keyTyped - натискане на произволен клавиш различен от (действие) стрелки, Home, End, Page Up, Page Down, функционален клавиш, Num Lock, Print Screen, Scroll Lock, Caps Lock и Pause
    - keyReleased– изпълнява се след keyPressed или keyTyped
  - Всеки от тези методи взима за аргумент KeyEvent

## Outline

KeyDemoFrame  
.java

(1 от 3)

```

1 // Fig. 11.36: KeyDemoFrame.java
2 // Demonstrating keystroke events.
3 import java.awt.Color;
4 import java.awt.event.KeyListener;
5 import java.awt.event.KeyEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JTextArea;
8
9 public class KeyDemoFrame extends JFrame implements KeyListener
10 {
11     private String line1 = ""; // first line of textarea
12     private String line2 = ""; // second line of textarea
13     private String line3 = ""; // third line of textarea
14     private JTextArea textArea; // textarea to display output
15
16     // KeyDemoFrame constructor
17     public KeyDemoFrame()
18     {
19         super( "Demonstrating Keystroke Events" );
20
21         textArea = new JTextArea( 10, 15 ); // set up JTextArea
22         textArea.setText( "Press any key on the keyboard..." );
23         textArea.setEnabled( false ); // disable textarea
24         textArea.setDisabledTextColor( Color.BLACK ); // set text color
25         add( textArea ); // add textarea to JFrame
26
27         addKeyListener( this ); // allow frame to process key events
28     } // end KeyDemoFrame constructor
29

```

Прилага KeyListener interface

Задава цвят за фон

Регистрира приложението като  
обект за обработка на събитието



## Outline

Дефинира метод keyPressed

Взима кода на натиснат клавиш и извежда описание на клавиша

Дефинира метод keyReleased

Взима кода на освободен клавиш и извежда описание на клавиша

Дефинира метод keyTyped

KeyDemoFrame  
.java

(2 от 3)

Извежда натисната буква на клавиш

```

30 // handle press of any key
31 public void keyPressed( KeyEvent event )
32 {
33     line1 = String.format( "key pressed: %s",
34         event.getKeyText( event.getKeyCode() ) ); // output pressed key
35     setLines2and3( event ); // set output lines two and three
36 } // end method keyPressed
37
38 // handle release of any key
39 public void keyReleased( KeyEvent event )
40 {
41     line1 = String.format( "key released: %s",
42         event.getKeyText( event.getKeyCode() ) ); // output released key
43     setLines2and3( event ); // set output lines two and three
44 } // end method keyReleased
45
46 // handle press of an action key
47 public void keyTyped( KeyEvent event )
48 {
49     line1 = String.format( "key typed: %s", event.getKeyChar() );
50     setLines2and3( event ); // set output lines two and three
51 } // end method keyTyped
52

```



## Outline

```
53 // set second and third lines of output
54 private void setLines2and3( KeyEvent event )
55 {
56     line2 = String.format( "This key is %san action key",
57         ( event.isActionKey() ? "" : "not " ) );
58
59     String temp = event.getKeyModifiersText( event.getModifiers() );
60
61     line3 = String.format( "Modifier keys pressed: %s",
62         ( temp.equals( "" ) ? "none" : temp ) ); // output modifiers
63
64     textArea.setText( String.format( "%s\n%s\n%s\n",
65         line1, line2, line3 ) ); // output three lines of text
66 } // end method setLines2and3
67 } // end class KeyDemoFrame
```

Проверява дали е клавиш за действие

Определя модификаторите на клавиша

KeyDemoFrame  
.java

(3 от 3)



## Outline

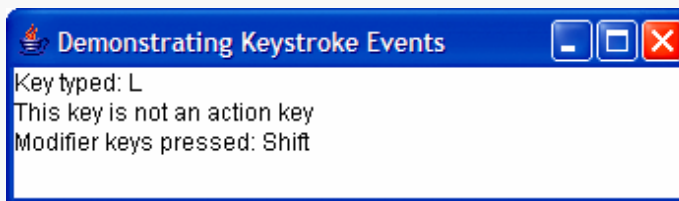
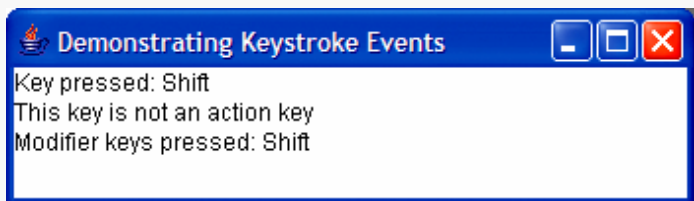
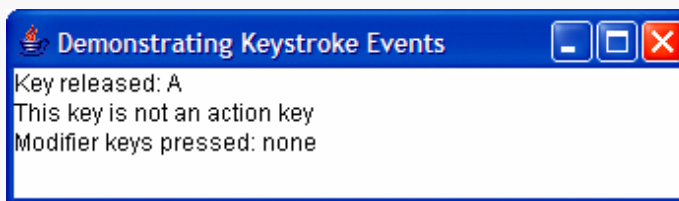
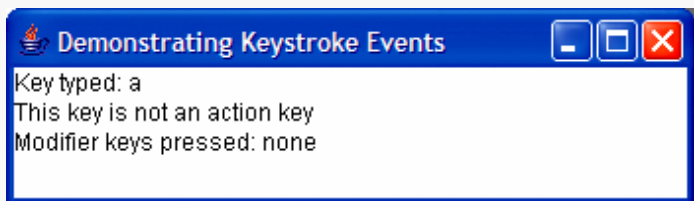
### KeyDemo.java

(1 от 2)

```

1 // Fig. 11.37: KeyDemo.java
2 // Testing KeyDemoFrame.
3 import javax.swing.JFrame;
4
5 public class KeyDemo
6 {
7     public static void main( String args[] )
8     {
9         KeyDemoFrame keyDemoFrame = new KeyDemoFrame();
10        keyDemoFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11        keyDemoFrame.setSize( 350, 100 ); // set frame size
12        keyDemoFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class KeyDemo

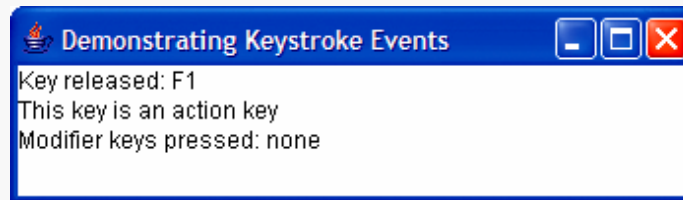
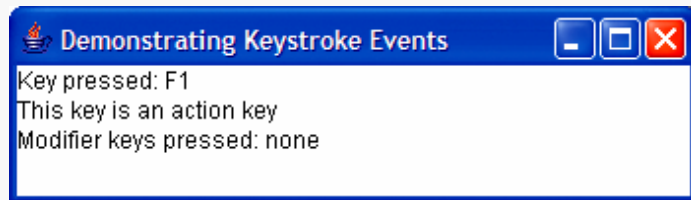
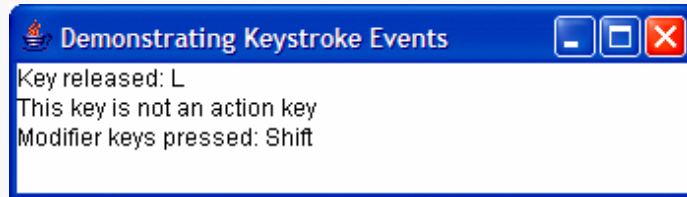
```



## Outline

### KeyDemo.java

(1 от 2)



# Задачи

1. Напишете Java *JFrame* приложение, което изобразява панел в средата, на който е изписан символа 'А' (писането на Текст върху графична компонента става с командата *drawString(x,y, stringToDraw)* на *Graphics* обект *g* по подобие на *drawOval()*, *drawRect()*, *drawLine()* и пр.) Нека при натиснат на бутон на мишката върху символа 'А' този символ да се мести заедно с курсора на мишката върху панела до отпускане на бутона на мишката.
2. Напишете Java *JFrame* приложение , което е вариант на *задача 1*, при който символа се мести нагоре, надолу, наляво и надясно в панела при използване на съответните стрелки от клавиатурата. Като упътване използвайте, че методът *getKeyCode()* на *KeyEvent* връща *KeyEvent.VK\_DOWN*, *KeyEvent.VK\_UP*, *KeyEvent.VK\_LEFT*, *KeyEvent.VK\_RIGHT* целочислени константи при натискане на стрелките нагоре, надолу, наляво и надясно.