ГРАФИЧНИ ИНТЕРФЕЙСИ

ЛЕКЦИОНЕН КУРС "ООП(JAVA)"

Distributed @learning center

УВОД

- Въпреки че програмите, използващи конзола, са отлични за преподаване на основите на Java, повечето приложения в реалния свят са GUI базирани
- Най-широко използваният Java GUI е Swing
- Swing дефинира колекция от класове и интерфейси, които поддържат богат набор от визуални компоненти
  - Като напр., бутони, текстови полета, падащи панели, чек-боксове, дървета, ...
- Съвместно използвани с тези контроли могат да се изграждат мощни и лесни за използване графични интерфейси

УВОД

- Важно е да се отбележи, че Swing е много голяма тема
  - Ще разгледаме базовите концепции и конкретни примери за използването й
- Трябва да споменем, че от JDK 8 е създаден нов GUI за Java
  - Наречен JavaFX
  - Осигурява мощен, рационализиран и гъвкав подход, който опростява създаването на визуално креативен GUI
  - Като такъв, JavaFX очевидно е позиционирана като платформа на бъдещето

ИСТОРИЧЕСКИ БЕЛЕЖКИ

- Swing не съществува в ранните дни на Java
- Възниква в отговор на решаване на недостатъците, присъстващи в оригиналния GUI на Java – AWT (Abstract Window Toolkit)
  - AWT определя основен набор от компоненти, които поддържат използваем, но ограничен графичен интерфейс
- Една от причините за ограничената природа на AWT е, че той транслира визуалните компоненти в техните съответни платформи-специфични еквиваленти
  - Това означава, че външният вид (the look and feel) на AWT компонентите се определя от платформата, а не от Java
- Започвайки с Java 2, Swing беше напълно интегрирана в Java

ОБЩА ХАРАКТЕРИСТИКА

- Swing адресира ограниченията, свързани с компонентите на AWT, чрез използването на две основни концепции:
  - Леки компоненти (lightweight components)
  - Сменяем LAF (pluggable look and feel)
- Макар до голяма степен прозрачни за програмиста, тези две характеристики са в основата на дизайнерската философия на Swing и причината за голяма част от нейната мощ и гъвкавост

ЛЕКИ КОМПОНЕНТИ

- С много малко изключения компонентите на Swing са леки
  - Това означава, че са написани изцяло в Java
  - Не разчитат на специфични платформи
- Леките компоненти имат някои важни предимства, включително ефективност и гъвкавост
  - Тъй като леките компоненти не се транслират в специфични за платформата еквиваленти, външният вид на всеки компонент се определя от Swing, а не от основната операционна система
    - Това означава, че всеки компонент може да работа във всички платформи
  - Възможно е да се отдели външния вид на компонента от логиката на компонента (и това прави Swing)

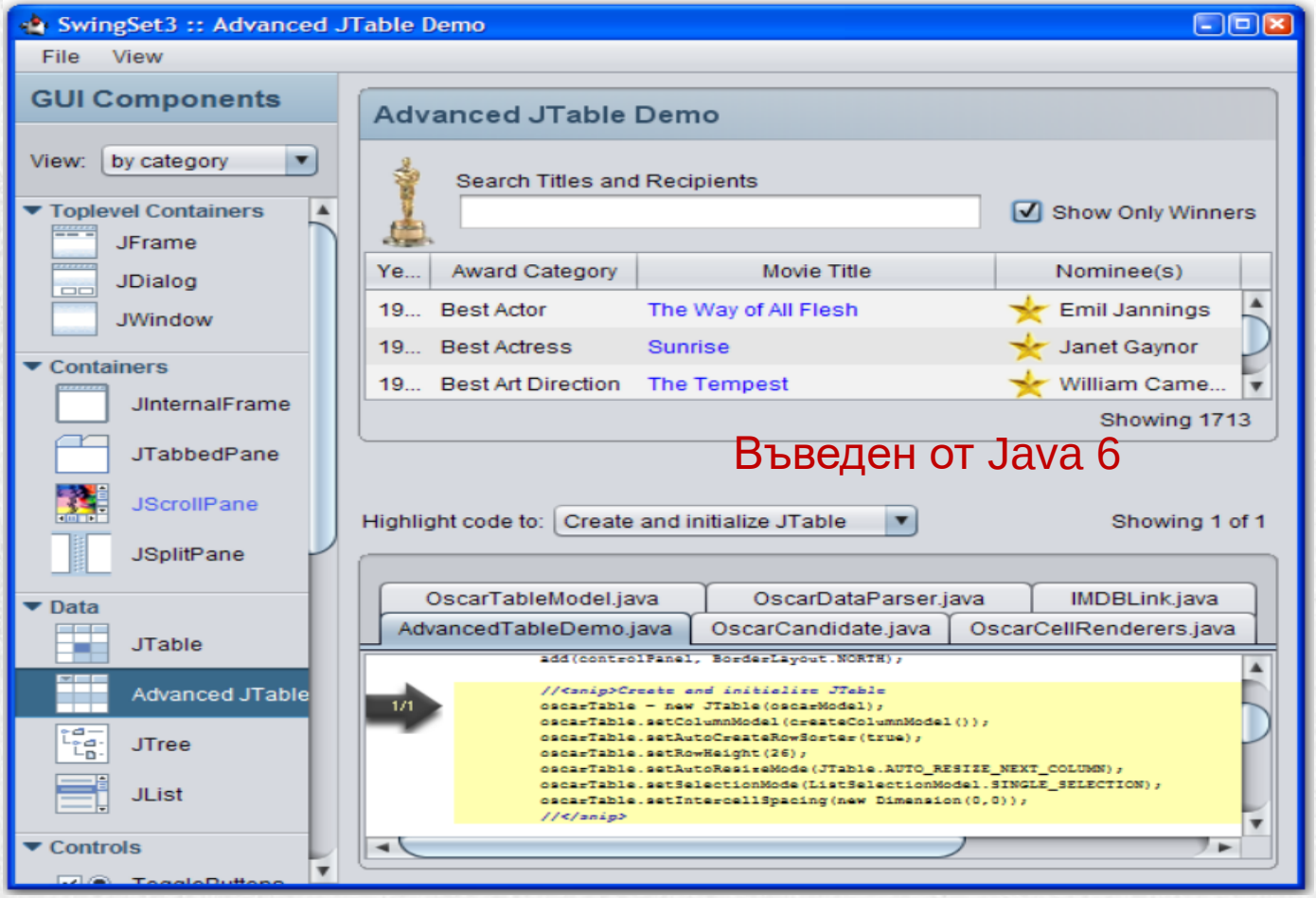
СМЕНЯЕМ LAF

- Тъй като всеки компонент на Swing се изобразява по-скоро от Java код, отколкото от платформени специфични еквиваленти, отделянето на външния вид и усещането (LAF) осигурява значително предимство
  - Става възможно да се промени начинът, по който даден компонент се възпроизвежда, без да се засягат някои от другите му аспекти
  - С други думи, е възможно да "включим" (plug in) нов външен вид и усещане за всеки даден компонент
    - Без странични ефекти в кода, който използва този компонент

LAF

- “Look And Feel” (LAF)
- Java предоставя различни LAF (напр., metal и Nimbus), които са достъпни за всички Swing потребители
- Metal LAF се нарича още Java LAF
  - Независим от платформата
  - Достъпен във всички имплементации на Java среди
  - По подразбиране LAF
  - Използва се от примерите

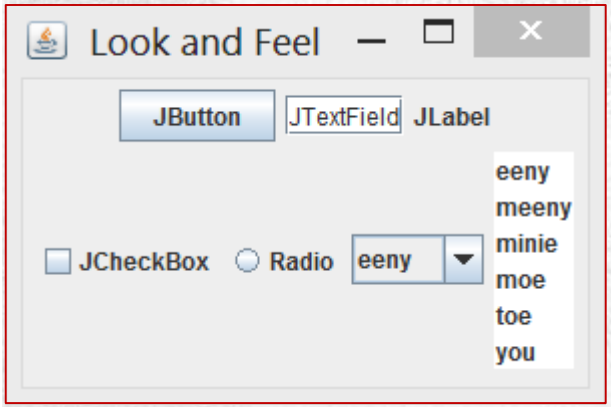
NIMBUS LOOK-AND-FEEL



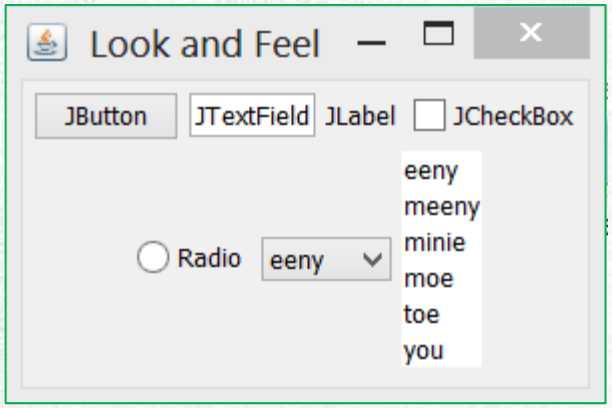
Въведен от Java 6

ДРУГИ LAF

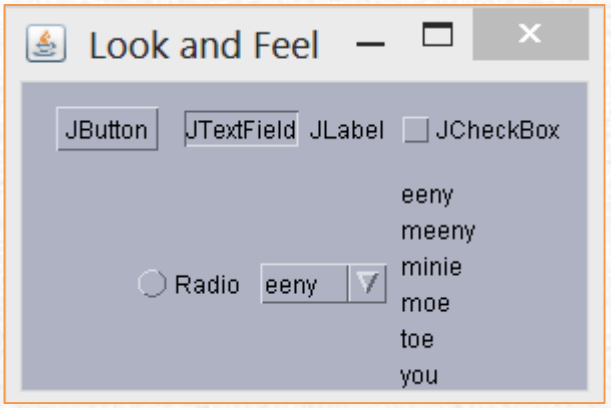
cross



system



motif



MVC

- Завършеният LAF на Swing е възможен, защото Swing използва модифицирана версия на класическата архитектура на модел-изглед-контролер (MVC)
- В терминологията на MVC:
  - Моделът кореспондира с информация за състоянието, свързана с компонента
    - Напр., в случай на чек-бокс, моделът съдържа поле, което показва дали полето е отметнато или не е отметнато
  - Изгледът определя как компонентът се показва на екрана, включително всички аспекти на изгледа, които са засегнати от текущото състояние на модела
  - Контролерът определя начина, по който компонентът реагира
    - Напр., когато потребителят кликне върху чек-бокса, контролерът реагира, като промени модела, за да отрази избора на потребителя (отметнат или не)
- Чрез разделянето на един компонент на модел, изглед и контролер, специфичното изпълнение на всеки от тях може да бъде променено, без да се засягат другите две

БАЗОВИ ЕЛЕМЕНТИ

- Swing GUI се състои от два ключови елемента:
  - Компонент
  - Контейнер
- Това разграничение е концептуално
  - Всички контейнери също са компоненти
- Разликата между двете се намира в предназначението им:
  - Компонентът е независим визуален контрол
  - Напр., бутон или текстово поле

КОНТЕЙНЕРИ

- Контейнерът съдържа група компоненти
  - Специален тип компонент, предназначен да съдържа други компоненти
- За да се визуализира един компонент, той трябва да се съдържа в контейнер
  - Така, всички Swing компоненти имат поне един контейнер
- Тъй като контейнерите са компоненти, контейнерът може да съдържа и други контейнери
  - Това позволява на Swing да дефинира контейнерна йерархия
  - Корен трябва да е контейнер от най-високо ниво

КОМПОНЕНТИ

- По принцип компонентите на Swing се наследяват от класа JComponent
  - Единствено изключение - четири контейнери от най-високо ниво
- JComponent осигурява функционалност, обща за всички компоненти
  - JComponent наследява контейнерите и компонентите на AWT класовете
  - По този начин компонент "Swing" е изграден и съвместим с компонент AWT
- Всички компоненти на Swing са представени от класовете на пакета javax.swing
  - Всички класове на компоненти започват с “J”

ПРЕГЛЕД НА КОМПОНЕНТИТЕ

JApplet (deprecated by JDK 9)	JButton	JCheckBox	JCheckBoxMenuItem
JColorChooser	JComboBox	JComponent	JDesktopPane
JDialog	JEditorPane	JFileChooser	JFormattedTextField
JFrame	JInternalFrame	JLabel	JLayer
JLayeredPane	JList	JMenu	JMenuBar
JMenuItem	JOptionPane	JPanel	JPasswordField
JPopupMenu	JProgressBar	JRadioButton	JRadioButtonMenuItem
JRootPane	JScrollBar	JScrollPane	JSeparator
JSlider	JSpinner	JSplitPane	JTabbedPane
JTable	JTextArea	JTextField	JTextPane
JTogglebutton	JToolBar	JToolTip	JTree
JViewport	JWindow		

ВИДОВЕ КОНТЕЙНЕРИ

- Swing дефинира два типа контейнери:
  - Контейнери от най-високо ниво
    - JFrame
    - JApplet (отхвърлен от JDK 9)
    - JWindow
    - JDialog
  - Леки контейнери
    - Напр., JPanel, JScrollPane, JRootPane

КОНТЕЙНЕРИ ОТ НАЙ-ВИСОКО НИВО

- Тези контейнери не наследяват JComponent
  - Наследяват AWT класовете Component и Container
- Контейнерите от най-високо ниво са тежки
  - Това прави контейнерите от най-високо ниво специален случай в библиотеката на Swing компонентите
- Както подсказва името, контейнерът от най-високо ниво трябва да е в горната част на контейнерната йерархия
  - Всяка йерархия трябва да започне с контейнер от най-високо ниво
  - Контейнер от първо ниво не се съдържа в друг контейнер
  - Един от най-често използваните е JFrame

ЛЕКИ КОНТЕЙНЕРИ

- Наследяват JComponent
- Леките контейнери често се използват за колективно организиране и управление на групи от свързани компоненти
- Един лек контейнер може да се съхранява в друг контейнер
  - По този начин могат да се използват за създаване на подгрупи от свързани контроли, които се съдържат във външен контейнер

КОНТЕЙНЕРНИ ПАНЕЛИ ОТ НАЙ-ВИСОКО НИВО

- Всеки контейнер от най-високо ниво определя набор от панели
  - В горната част на йерархията е инстанция на JRootPane
- JRootPane е лек контейнер за управление на други панели
  - Също така помага за управлението на опционната лента с менюта
- Възможни панели-корени:
  - Стъклен панел (glass pane)
  - Панел за съдържание (content pane)
  - Слоест панел (layered pane)

СТЪКЛЕН ПАНЕЛ

- Стъкленият панел е най-високото ниво
  - Най-високо и напълно покрива всички останали панели
- Позволява управление на събитията на мишката, които засягат целия контейнер (а не индивидуален контрол)
  - Също напр., оцветяване върху всеки друг компонент
- В повечето случаи не се налага директно използване на стъкления панел

СЛОЕСТ ПАНЕЛ

- Слоестите панели позволяват разполагане на компонентите в дълбочина
  - Също препокриване на компоненти
- Съдържа панела за съдържание и (опционално) лента с менюта
- Макар че стъкленият и слоестият панели са неразделна част от работата на контейнера от най-високо ниво и обслужат важни цели, голяма част от това, което те осигуряват се случва "зад сцената"

ПАНЕЛ ЗА СЪДЪРЖАНИЕ

- Панелът, с който приложенията взаимодействат най-много, е панелът за съдържание
  - Това е панелът, на който добавяме визуални компоненти
- С други думи, когато добавим компонент (напр. бутон) към контейнер от най-високо ниво, добавяме го към панела за съдържание
  - Поради това в панела за съдържание се съхраняват компонентите, с които потребителят взаимодейства

МЕНИДЖЪРИ НА ОФОРМЛЕНИЕТО

- Мениджърът на оформлението управлява позицията на компонентите в контейнера
- Java предлага няколко мениджъри за оформление
  - Повечето са осигурени от AWT (в рамките на java.awt), но Swing добавя няколко свои собствени
  - Всички мениджъри на оформлението са обекти от клас, който имплементира интерфейса LayoutManager
    - Някои също така имплементират интерфейса LayoutManager2

ПРЕГЛЕД НА МЕНИДЖЪРИ НА ОФОРМЛЕНИЕ

FlowLayout	A simple layout that positions components left-to-right, top-to-bottom. (Positions components right-to-left for some cultural settings.)
BorderLayout	Positions components within the center or the borders of the container. This is the default layout for a content pane.
GridLayout	Lays out components within a grid.
GridBagLayout	Lays out different size components within a flexible grid.
BoxLayout	Lays out components vertically or horizontally within a box.
SpringLayout	Lays out components subject to a set of constraints.

ДВА МЕНИДЖЪРА

- Двама мениджъри за оформление, които са много лесни за използване:
  - BorderLayout
  - FlowLayout

BORDERLAYOUT

- BorderLayout е мениджърът на оформлението по подразбиране за панела за съдържание
- Изпълнява стил на оформление, който дефинира пет местоположения, към които може да се добави компонент
  - Център (по подразбиране)
  - Север
  - Юг
  - Изток
  - Запад
- За да добавите компонент към един от другите региони се посочва името му

FLOWLAYOUT

- FlowLayout определя компонентите по редове, отгоре надолу
- Когато един ред е пълен, оформлението преминава към следващия ред
- Схемата доставя ограничен контрол върху разположението на компонентите, но е лесна за използване
- Ако променим размера на рамката, позицията на компонентите ще се промени

ПРИМЕР

```
import javax.swing.*;
public class SwingDemo {
        SwingDemo() {
            // Create a new JFrame container.
            JFrame jfrm = new JFrame("A Simple Swing Application");
           // Give the the frame an initial size.
            jfrm.setSize(275, 100);
           // Terminate the program when the user closes the application.
           jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           // Create a text-based label.
           JLabel jlab = new JLabel(" GUI programing with Swing.");
           // Add the label to the content pane.
           jfrm.add(jlab);
          // Display the frame.
          jfrm.setVisible(true);
      }
```

```
public static void main(String[] args) {
        // Create the frame on the event dispatching thread.
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new SwingDemo();
            }
         });
    }
}
```

ПРИМЕР

```
import javax.swing.*;
public class SwingDemo {
        SwingDemo() {
            // Create a new JFrame contai
            JFrame jfrm = new JFrame("A S
            // Give the the frame an initial
            jfrm.setSize(275, 100);
            // Terminate the program whei
            jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            // Create a text-based label.
            JLabel jlab = new JLabel(" GUI programing with Swing.");
            // Add the label to the content pane.
            jfrm.add(jlab);
            // Display the frame.
            jfrm.setVisible(true);
        }
```

Swing програмите трябва да импортират javax.swing, който съдържа компонентите и моделите, дефинирани от Swing.
Напр., класове за етикети, бутони, контроли за редактиране и менюта.
Пакетът трябва да бъде включен във всички програми, използващи Swing.

```
public static void main(String[] args) {
        // Create the frame on the event dispatching thread.
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new SwingDemo();
            }
        });
    }
}
```

ПРИМЕР

```
import javax.swing.*;
public class SwingDemo {
        SwingDemo() {
           // Create a new JFrame con...
           JFrame jfrm = new JFrame("A Simple Swing Application");
           // Give the the frame an initial size.
           jfrm.setSize(275, 100);
           // Terminate the program when the user closes the application.
           jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           // Create a text-based label.
           JLabel jlab = new JLabel(" GUI programing with Swing.");
           // Add the label to the content pane.
           jfrm.add(jlab);
           // Display the frame.
           jfrm.setVisible(true);
      }
```

- Конструкторът е мястото, където се извършва по-голямата част от действието на програмата.
- Започва със създаване на JFrame

```
public static void main(String[] args) {
        // Create the frame on the event dispatching thread.
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new SwingDemo();
            }
        });
    }
}
```

ПРИМЕР

```
import javax.swing.*;
public class SwingDemo {
        SwingDemo() {
          // Create a new JFrame container.
          JFrame jfrm = new JFrame("A Simple Swing Application");
          // Give the the frame an initial size.
          jfrm.setSize(275, 100);
          // Terminate the program when
          jfrm.setDefaultCloseOperation(JF
          // Create a text-based label.
          JLabel jlab = new JLabel(" GUI pr
          // Add the label to the content p
          jfrm.add(jlab);
          // Display the frame.
          jfrm.setVisible(true);
    }
```

```
pu
                public void run() {
                    new SwingDemo();
                }
            });
    }
}
```

- Създава контейнер, наречен jfrm, който дефинира правоъгълния прозорец, съдържащ заглавна лента и бутони за затваряне, минимизиране, максимизиране и възстановяване на бутоните, както и системно меню.
- Това е стандартен прозорец от първо ниво. Заглавието на прозореца се предава на конструктора.

ПРИМЕР

```
import javax.swing.*;
public class SwingDemo {
        SwingDemo() {
           // Create a new JFrame container.
           JFrame jfrm = new JFrame("A Simple Swing Application");
           // Give the the frame an initial size.
           jfrm.setSize(275, 100);
           // Terminate the program when the user closes the application.
           jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           // Create a text-based label
```

- По подразбиране, когато се затвори прозорец от най-високо ниво (напр., потребителят кликне върху close box), прозорецът се премахва от екрана, но приложението не е прекратено.
- Това поведение (по подразбиране) е полезно в някои ситуации – в много случаи искаме цялото приложение да завърши, когато неговият прозорец от най-високо ниво е затворен.
- Има няколко начина за постигане това.
- Най-лесният начин е използване на метода setDefaultCloseOperation () (както прави програмата) – аргументът определя какво се случва, когато прозорецът е затворен.

```
}
```

ПРИМЕР

```
import javax.swing.*;
public class SwingDemo {
        SwingDemo() {
            // Create a new JFrame container.
            JFrame jfrm = new JFrame("A Simple Swing Application");
            // Give the the frame an initial size.
            jfrm.setSize(275, 100);
            // Terminate the program when the user closes the applicatio
            jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            // Create a text-based label.
            JLabel jlab = new JLabel(" GUI programing with Swing.");
            // Add the label to the content pane.
            jfrm.add(jlab);
            // Display the frame.
            jfrm.setVisible(true);
    }
```

- Създава JLabel компонент
- Просто показва текст, икона или комбинация от двете
- Тук етикетът съдържа само текст, предаден на конструктора

```
public static void main(String[] args) {
        // Create the frame on the event dispatching thread.
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new SwingDemo();
            }
        });
    }
}
```

ПРИМЕР

```
import javax.swing.*;
public class SwingDemo {
        SwingDemo() {
            // Create a new JFrame cc
            JFrame jfrm = new JFrame
            // Give the the frame an ir
            jfrm.setSize(275, 100);
            // Terminate the program
            jfrm.setDefaultCloseOperat
            // Create a text-based label
            JLabel jlab = new JLabel(" C
            // Add the label to the con
            jfrm.add(jlab);
            // Display the frame.
            jfrm.setVisible(true);
    }
}
```

- Добавя етикета към панела за съдържание на рамката
- Всички контейнери от най-високо ниво имат панел за съдържание, в който се съдържат компонентите
- За добавяне на компонент към рамка, трябва да го добавим панела за съдържание на рамката чрез add() метода
- Методът add () има няколко версии:
  - По подразбиране панелът за съдържание, свързан с JFrame, използва оформление на границата – тази версията на add () добавя компонента (в този случай етикет) в центъра и неговият размер автоматично се настройва така, че да съответства на размера на центъра
  - Други версии на add () позволяват да посочим един от граничните региони

ПРИМЕР

```
import javax.swing.*;
public class SwingDemo {
        SwingDemo() {
           // Create a new JFrame container.
           JFrame jfrm = new JFrame("A Simple Swing Application");
          // Give the the frame an initial size.
          jfrm.setSize(275, 100);
          // Terminate the program when the user closes the application.
          jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          // Create a text-based label.
          JLabel jlab = new JLabel(" GUI programing with Swing.");
          // Add the label to the content pane.
          jfrm.add(jlab);
         // Display the frame.
         jfrm.setVisible(true);
    }
```

Последният оператор в конструктора прави прозореца да стане видим

```
public static void main(String[] args) {
        // Create the frame on the event dispatching thread.
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new SwingDemo();
            }
         });
    }
}
```

ПРИМЕР

```
import javax.swing.*;
public class SwingDemo {
        SwingDemo() {
            // Create a new JFrame container.
            JFrame jfrm = new JFrame("A Simple Swing Applicatio
            // Give the the frame an initial size.
            jfrm.setSize(275, 100);
            // Terminate the program when the user closes the appl
            jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE
            // Create a text-based label.
            JLabel jlab = new JLabel(" GUI programing with Swing.")
            // Add the label to the content pane.
            jfrm.add(jlab);
            // Display the frame.
            jfrm.setVisible(true);
    }
```

- В main() метода се създава се обект SwingDemo, който предизвиква прозорецът и етикетът да се покажат
- Конструкторът на SwingDemo се използва по показания начин
- Създаването на обекта става в нишка на диспечиране на събития, а не в основната нишка на приложението

```
public static void main(String[] args) {
        // Create the frame on the event dispatching thread.
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new SwingDemo();
            }
        });
    }
}
```

СЪБИТИЯ

- Най-общо, Swing програмите се направляват от събития
- Когато потребителят взаимодейства с един компонент, се генерира събитие
- Събитието се предава на приложението, като се извиква манипулатор на събития, определен от приложението
- Обаче манипулаторът се изпълнява на нишката на диспечера на събития, осигурен от Swing
  - А не на основната нишка на приложението
- По този начин, въпреки че манипулаторите на събития се определят от нашата програма, те се изпълняват на нишка, която не е създадена от нашата програма

СЪБИТИЯ

- За да се избегнат проблеми - напр., две различни нишки, опитващи се да обновяват един и същ компонент едновременно - всички компоненти на Swing GUI трябва да бъдат създадени и актуализирани от нишката на диспечер на събития, а не основната нишка на приложението
- Обаче, main() се изпълнява на основна нишка
  - По този начин тя не може директно да инстанцира един SwingDemo обект
  - Вместо това, трябва да създаде Runnable (изпълним) обект, изпълняващ се на нишката на диспечера на събития
  - Този обект да създава GUI

СЪБИТИЯ

- За да се направи възможно създаване кода на графичния интерфейс (GUI) на диспечерската нишка, трябва да използваме един от двата метода, определени от класа SwingUtilities:
  - invokeLater()
  - invokeAndWait()
- Общ вид:
  - static void invokeLater (Runnable obj)
  - static void invokeAndWait (Runnable obj) throws InterruptedException, InvocationTargetException

СЪБИТИЯ

- obj е Runnable обект, чийто run() метод се извиква от диспечерската нишка
- Разликата между двата метода:
  - invokeLater() - run() се изпълнява асинхронно с диспечерската AWT нишка на събитието
  - invokeAndWait() - run() се изпълнява синхронно с диспечерската AWT нишка на събитието
- Можем да използваме тези методи за извикване на метод, който изгражда графичния интерфейс за Swing приложението
  - Или когато е необходимо да променим състоянието на GUI от код не изпълняван от нишката на диспечера на събития

СЪБИТИЯ

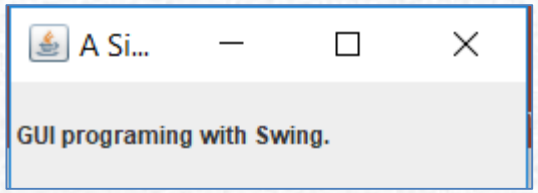
- Програмата от примера не отговаря на никакви събития
  - Понеже JLabel е пасивен компонент
  - Т.е., JLabel не генерира събития
- Всички останали компоненти обаче генерират събития, на които програмите трябва да реагират

ПРИМЕР

```
import javax.swing.*;
public class SwingDemo {
        SwingDemo() {
           // Create a new JFrame container.
           JFrame jfrm = new JFrame("A Simple Swing Application");
           // Give the the frame an initial size.
           jfrm.setSize(275, 100);
           // Terminate the program when the user closes the application.
           jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           // Create a text-based label.
           JLabel jlab = new JLabel(" GUI programing with Swing.");
           // Add the label to the content pane.
           jfrm.add(jlab);
           // Display the frame.
           jfrm.setVisible(true);
      }
```

```
public static void main(String[] args) {
        // Create the frame on the event dispatching thread.
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new SwingDemo();
            }
         });
    }
}
```

? Резултат



ОБРАБОТКА НА SWING СЪБИТИЯ

- Принципно Swing програмите се управляват от събития, като компонентите взаимодействат с програмата чрез събития
  - Напр., едно събитие се генерира, когато потребителят кликне върху даден бутон, премества мишката, натиска клавиш или избира елемент от списък
- Събитията могат да се генерират и по друг начин
  - Напр., събитие се генерира, когато таймерът изтече
- Когато дадено събитие е изпратено на дадена програма, програмата отговаря на събитието, като използва манипулатор на събития
  - По този начин управлението на събитията е важна част от почти всички приложения на Swing

СЪБИТИЕН МОДЕЛ

- Механизмът за управление на събития, използван от Swing, се нарича модел на делигиране на събития (delegation event model)
- Нейната концепция е съвсем проста
  - Един източник генерира събитие и го изпраща на един или повече слушатели
  - При този подход слушателят просто чака, докато не получи събитие
  - След като събитието пристигне, слушателят обработва събитието и връща

СЪБИТИЕН МОДЕЛ

- Предимството на този подход е, че логиката на приложението, която обработва събитията, е чисто отделена от логиката на потребителския интерфейс, който генерира събитията
- Следователно елементът на потребителския интерфейс е в състояние да "делегира" обработката на събитие на отделена част от кода
- В събитийния модел слушателят трябва да се регистрира в източника за да може да получава едно събитие

СЪБИТИЯ

- В Java едно събитие е обект, който описва промяна в състоянието на един източник на събития
- Може да бъде генерирано като последица от:
  - Взаимодействие на потребител с елемент от графичния потребителски интерфейс или
  - Под програмен контрол
- Суперкласът за всички събития е java.util.EventObject
  - Много събития са декларирани в java.awt.event
  - Събития, специално свързани с Swing, се намират в javax.swing.event

ИЗТОЧНИК НА СЪБИТИЕ

- Източник на събитие е обект, който генерира събитие
- Когато източникът генерира едно събитие, той го изпраща на всички регистрирани слушатели
- По тази причина, за да получават събития, слушателите трябва да се регистрира в източника
- В Swing, слушателите се регистрират като извикат метод на обекта от тип източник
- Всеки тип събитие има свой собствен регистрационен метод
- Обикновено събитията използват следната конвенция:
  - public void addTypeListener (TypeListener el)

ПРИМЕРИ

- Напр.,:
  - Методът за регистриране слушател на събития от клавиатурата е addKeyListener()
  - Методът за регистрира слушател на движение на мишката е addMouseMotionListener()
- Когато се случи събитие, събитието се предава на всички регистрирани слушатели

ОТМЯНА РЕГИСТРАЦИЯ

- Източникът трябва също така да предостави метод, който позволява на слушателя да отмени интереса си към конкретен тип събитие
- В Swing, такъв метод е следният:
  - public void removeTypeListener (TypeListener el)

ПРЕДОСТАВЯНЕ НА МЕТОДИ

- Методите за добавяне или премахване на слушатели се предоставят от източника, генериращ събития
  - Напр., класът JButton е източник на ActionEvents, които са събития, които показват, че някои действия с бутона са станали
- По този начин JButton предоставя методи за добавяне или премахване на слушател

СЛУШАТЕЛИ НА СЪБИТИЯ

- Слушател е обект, който се идентифицира, когато възникне събитие
- Две основни изисквания:
  - Трябва да е регистриран в един или повече източници за получаване на определен тип събития
  - Трябва да имплементира метод за получаване и обработка на такива събития

ИНТЕРФЕЙСИ

- Методите, които получават и обработват събития, приложими за Swing, са дефинирани в набор от интерфейси
  - Като тези, намиращи се в java.awt.event и javax.swing.event
- Напр., интерфейсът ActionListener дефинира метод, който обработва един ActionEvent
  - Всеки обект може да получи и обработва това събитие, ако предоставя имплементация на интерфейса ActionListener

ОБЩ ПРИНЦИП

- Съществува важен общ принцип
  - Обработващият едно събитие трябва да свърши своята работа бързо и след това да се върне
  - В повечето случаи не трябва да се занимава с дълга операция тъй като това ще забави цялото приложение
    - Ако е необходима операция, изискваща повече време, е добре да се създаде отделна нишка за нея

СЪБИТИЙНИ КЛАСОВЕ И ИНТЕРФЕЙСИ НА СЛУШАТЕЛИ

- Класовете на събития са в основата на обработката на събитията от Swing механизма
- Коренът на йерархията на събитийните класове е EventObject от java.util
  - Суперклас на всички събития в Java
- Класът AWTEvent от пакета java.awt е подклас на EventObject
- Той е суперкласът (пряко или непряко) на всички базирани на AWT събития, използвани в модела на събития
- Въпреки че Swing използва AWT събитията, той добавя и няколко свои собствени
  - Те са в javax.swing.event
- Swing поддържа голям брой събития

ПРЕГЛЕД НА КЛАСОВЕ И ИНТЕРФЕЙСИ

Event Class	Description	Corresponding Event Listener
ActionEvent	Generated when an action occurs within a control, such as when a button is clicked.	ActionListener
ItemEvent	Generated when an item is selected, such as when a check box is clicked.	ItemListener
ListSelectionEvent	Generated when a list selection changes.	ListSelectionListener

JBUTTON

- Един от най-често използваните контроли в Swing е бутонът Jbutton
- JButton наследява абстрактния клас AbstractButton, дефиниращ функционалността, обща за всички бутони
- Swing бутоните могат да съдържат текст, изображение или и двете
- JButton доставя няколко конструктора
  - Напр., JButton (String msg), където msg определя низа, който ще се показва в бутона

JBUTTON

- При натискане на бутон той генерира ActionEvent
- JButton предоставя следните методи, които се използват за добавяне или премахване на слушатели:
  - void addActionListener(ActionListener *al*)
  - void removeActionListener(Action
- al определя обект, който получава известия за събития
- Този обект трябва да бъде инстанция на класа, който имплементира интерфейса ActionListener

СЪБИТИЕН МЕТОД

- Интерфейсът ActionListener дефинира само един метод: actionPerformed()
  - Този метод се извиква, когато се натисне бутона
  - Това е манипулаторът на събитие, настъпило при натискане на бутона
  - Имплементацията на actionPerformed () трябва бързо да отговори на това събитие и да се върне
  - Както беше обяснено по-рано манипулаторът на събития не трябва да се занимава с дълги операции, защото това ще забави цялото приложение

КОМАНДА ЗА ДЕЙСТВИЕ

- От обекта ActionEvent, предаден на actionPerformed(), можем да получим полезна информация за събитието, възникнало при натискане на бутон
  - Напр., команда за действие (action command), свързана с бутона
  - По подразбиране, това е низът, който се показва в бутона
  - Командата за действие се получава при извикване на метода getActionCommand() за обекта на събитието
- Командата за действие идентифицира бутона
  - Когато използваме два или повече бутона в едно и също приложение командата за действие дава лесен начин да определим кой бутон е натиснат

ПРИМЕР

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

public class ButtonDemo implements ActionListener {
    JLabel jlab;
    ButtonDemo() {
        JFrame jfrm = new JFrame("A Button Example");
        jfrm.setLayout(new FlowLayout());
        jfrm.setSize(220, 90);
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Make two buttons.
       JButton jbtnUp = new JButton("Up");
       JButton jbtnDown = new JButton("Down");
       // Add action listeners.
       jbtnUp.addActionListener(this);
       jbtnDown.addActionListener(this);
       // Add the buttons to the content plane.
       jfrm.add(jbtnUp);
       jfrm.add(jbtnDown);
       jlab = new JLabel("Press a button.");
       jfrm.add(jlab);
       jfrm.setVisible(true);
   }
    public void actionPerformed(ActionEvent ae) {
       if(ae.getActionCommand().equals("Up"))
          jlab.setText("You pressed Up.");
       else
          jlab.setText("You pressed Down.");
    }

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
       public void run() {
          new ButtonDemo();
       }
    });
  }

}
```

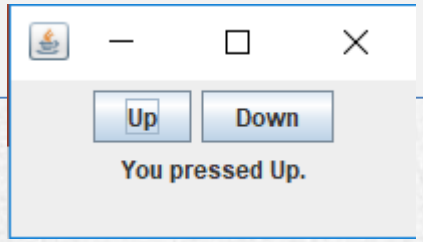
ПРИМЕР

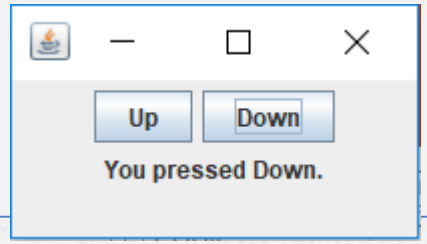
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ButtonDemo implements ActionListener {
    JLabel jlab;
    ButtonDemo() {
        JFrame jfrm = new JFrame("A Button Example");
        jfrm.setLayout(new FlowLayout());
        jfrm.setSize(220, 90);
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Make two buttons.
       JButton jbtnUp = new JButton("Up");
       JButton jbtnDown = new JButton("Down");
       // Add action listeners.
       jbtnUp.addActionListener(this);
       jbtnDown.addActionListener(this);
       // Add the buttons to the content plane.
       jfrm.add(jbtnUp);
       jfrm.add(jbtnDown);
       jlab = new JLabel("Press a button.");
       jfrm.add(jlab);
       jfrm.setVisible(true);
    }
    public void actionPerformed(ActionEvent ae) {
       if(ae.getActionCommand().equals("Up"))
          jlab.setText("You pressed Up.");
       else
          jlab.setText("You pressed Down.");
    }

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
       public void run() {
          new ButtonDemo();
       }
    });
 }

}
```



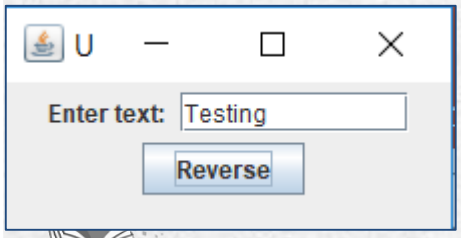


ПРИМЕР

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TFDemo implements ActionListener {
    JTextField jtf;
    JButton jbtnRev;
    JLabel jlabPrompt, jlabContents;
    TFDemo() {
      JFrame jfrm = new JFrame("Use a text field");
      jfrm.setLayout(new FlowLayout());
      jfrm.setSize(240, 120);
      jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      jtf = new JTextField(10);
      jtf.setActionCommand("myTF");
      JButton jbtnRev = new JButton("Reverse");
      jbtnRev.addActionListener(this);
      jtf.addActionListener(this);
      jlabPrompt = new JLabel("Enter text: ");
      jlabContents = new JLabel("");
      // Add the buttons to the content plane.
      jfrm.add(jlabPrompt);
      jfrm.add(jtf);
      jfrm.add(jbtnRev);
      jfrm.add(jlabContents);
      jfrm.setVisible(true);
    }
    public void actionPerformed(ActionEvent ae) {
      if(ae.getActionCommand().equals("Reverse")) {
        String orgStr = jtf.getText();
        String resStr = "";
        for(int i = orgStr.length() - 1; i >= 0; i--)
          resStr += orgStr.charAt(i);
          jtf.setText(resStr);
        } else
          jlabContents.setText("You pressed Enter. Text is: " +
                       jtf.getText());
    }
    public static void main(String[] args) {
      // Create the frame on the event dispatching thread.
      SwingUtilities.invokeLater(new Runnable() {
        public void run() {
          new TFDemo();
        }
      });
    }
}
```

ПРИМЕР

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TFDemo implements ActionListener {
    JTextField jtf;
    JButton jbtnRev;
    JLabel jlabPrompt, jlabContents;
    TFDemo() {
      JFrame jfrm = new JFrame("Use a text field");
      jfrm.setLayout(new FlowLayout());
      jfrm.setSize(240, 120);
      jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      jtf = new JTextField(10);
      jtf.setActionCommand("myTF");
      JButton jbtnRev = new JButton("Reverse");
      jbtnRev.addActionListener(this);
      jtf.addActionListener(this);
      jlabPrompt = new JLabel("Enter text: ");
      jlabContents = new JLabel("");
```




```
      // Add the buttons to the content plane.
      jfrm.add(jlabPrompt);
      jfrm.add(jtf);
      jfrm.add(jbtnRev);
      jfrm.add(jlabContents);
      jfrm.setVisible(true);
    }
    public void actionPerformed(ActionEvent ae) {
      if(ae.getActionCommand().equals("Reverse")) {
        String orgStr = jtf.getText();
        String resStr = "";
        for(int i = orgStr.length() - 1; i >= 0; i--)
          resStr += orgStr.charAt(i);
          jtf.setText(resStr);
        } else
          jlabContents.setText("You pressed Enter. Text is: " +
                    jtf.getText());
      }
      public static void main(String[] args) {
        // Create the frame on the event dispatching thread.
        SwingUtilities.invokeLater(new Runnable() {
          public void run() {
            new TFDemo();
          }
        });
      }
}
```

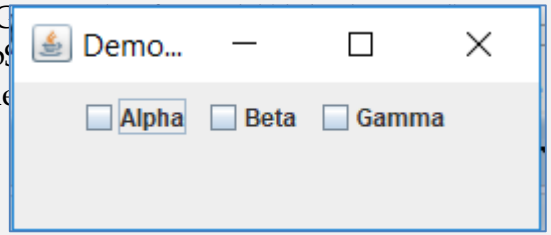
ПРИМЕР

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CBDemo implements ItemListener {
    JCheckBox jcbAlpha, jcbBeta, jcbGamma;
    JLabel jlabSelected, jlabChanged;
    CBDemo() {
      JFrame jfrm = new JFrame("Demonstare Check Boxes");
      jfrm.setLayout(new FlowLayout());
      jfrm.setSize(280, 120);
      jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOS
      jlabSelected = new JLabel("");
      jlabChanged = new JLabel("");
      jcbAlpha = new JCheckBox("Alpha");
      jcbBeta = new JCheckBox("Beta");
      jcbGamma = new JCheckBox("Gamma");
      jcbAlpha.addItemListener(this);
      jcbBeta.addItemListener(this);
      jcbGamma.addItemListener(this);
      jfrm.add(jcbAlpha);  jfrm.add(jcbBeta);
      jfrm.add(jcbGamma);  jfrm.add(jlabChanged);
      jfrm.add(jlabSelected);
     jfrm.setVisible(true);
}
```

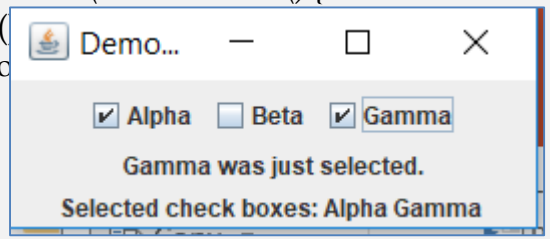
```
    public void itemStateChanged(ItemEvent ie) {
        String str = "";
        JCheckBox cb = (JCheckBox) ie.getItem();
        if(cb.isSelected())
          jlabChanged.setText(cb.getText() + " was just selected.");
        else
          jlabChanged.setText(cb.getText() + " was just cleared.");
        if(jcbAlpha.isSelected()) {
          str += "Alpha ";
        }
        if(jcbBeta.isSelected()) {
          str += "Beta ";
        }
        if(jcbGamma.isSelected()) {
          str += "Gamma ";
        }
        jlabSelected.setText("Selected check boxes: " + str);
    }
    public static void main(String[] args) {
        // Create the frame on the event dispatching thread.
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new CBDemo();
            }
        });
    }
}
```

ПРИМЕР

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CBDemo implements ItemListener {
    JCheckBox jcbAlpha, jcbBeta, jcbGamma;
    JLabel jlabSelected, jlabChanged;
    CBDemo() {
      JFrame jfrm = new JFrame("Demonstare Check Boxes");
      jfrm.setLayout(new FlowLayout());
      jfrm.setSize(280, 120);
      jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOS
      jlabSelected = new JLabel("");
      jlabChanged = new JLabel("");
      jcbAlpha = new JCheckBox("Alpha");
      jcbBeta = new JCheckBox("Beta");
      jcbGamma = new JCheckBox("Gamma");
      jcbAlpha.addItemListener(this);
      jcbBeta.addItemListener(this);
      jcbGamma.addItemListener(this);
      jfrm.add(jcbAlpha);  jfrm.add(jcbBeta);
      jfrm.add(jcbC
      jfrm.add(jlabS
      jfrm.setVisible
}
```




```
    public void itemStateChanged(ItemEvent ie) {
        String str = "";
        JCheckBox cb = (JCheckBox) ie.getItem();
        if(cb.isSelected())
          jlabChanged.setText(cb.getText() + " was just selected.");
        else
          jlabChanged.setText(cb.getText() + " was just cleared.");
        if(jcbAlpha.isSelected()) {
          str += "Alpha ";
        }
        if(jcbBeta.isSelected()) {
          str += "Beta ";
        }
        if(jcbGamma.isSelected()) {
          str += "Gamma ";
        }
        jlabSelected.setText("Selected check boxes: " + str);
    }
    public static void main(String[] args) {
        // Create the frame on the event dispatching thread.
        SwingUtilities.invokeLater(new Runnable() {
            public void run(
                new CBDemo
            }
        });
    }
}
```



ПРИМЕР

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class ListDemo implements ListSelectionListener {
   JList<String> jlst;
   JLabel jlab;
   JScrollPane jscrlp;
   String names[] = { "Sherry", "Jon", "Rachel",
                      "Sasha", "Josselyn", "Randy",
                      "Tom", "Mery", "Ken",
                      "Andrew", "Matt", "Todd" };
    ListDemo() {
       JFrame jfrm = new JFrame("JList Demo");
       jfrm.setLayout(new FlowLayout());
       jfrm.setSize(200, 160);
       jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       jlst = new JList<String>(names);
       jlst.setSelectionMode(ListSelectionModel.SINGLE_SELECTION
       jscrlp = new JScrollPane(jlst);
       jscrlp.setPreferredSize(new Dimension(120, 90));
       jlab = new JLabel("Please choose a name");
       jlst.addListSelectionListener(this);
       jfrm.add(jscrlp);
       jfrm.add(jlab);
       jfrm.setVisible(true);
}
```

```
   public void valueChanged(ListSelectionEvent le) {

      int idx = jlst.getSelectedIndex();

      if(idx != -1)
        jlab.setText("Current selection: " + names[idx]);
      else
        jlab.setText("Please choose a name");
   }

    public static void main(String[] args) {
    // Create the frame on the event dispatching thread.
    SwingUtilities.invokeLater(new Runnable() {
       public void run() {
         new ListDemo();
       }
    });
  }
}
```

ПРИМЕР

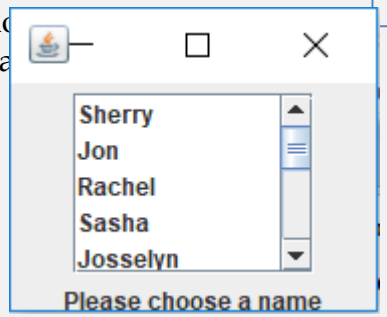
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class ListDemo implements ListSelectionListener {
   JList<String> jlst;
   JLabel jlab;
   JScrollPane jscrlp;
   String names[] = { "Sherry", "Jon", "Rachel",
                      "Sasha", "Josselyn", "Randy",
                      "Tom", "Mery", "Ken",
                      "Andrew", "Matt", "Todd" };
    ListDemo() {
       JFrame jfrm = new JFrame("JList Demo");
       jfrm.setLayout(new FlowLayout());
       jfrm.setSize(200, 160);
       jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       jlst = new JList<String>(names);
       jlst.setSelectionMode(ListSelectionModel.SINGLE_SELECTION
       jscrlp = new JScrollPane(jlst);
       jscrlp.setPreferredSize(new Dimensi
       jlab = new JLabel("Please choose a na
       jlst.addListSelectionListener(this);
       jfrm.add(jscrlp);
       jfrm.add(jlab);
       jfrm.setVisible(true);
}
```

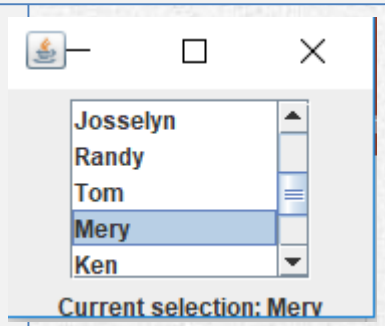
```
   public void valueChanged(ListSelectionEvent le) {

      int idx = jlst.getSelectedIndex();

      if(idx != -1)
        jlab.setText("Current selection: " + names[idx]);
      else
        jlab.setText("Please choose a name");
   }

    public static void main(String[] args) {
    // Create the frame on the event dispatching thread.
    SwingUtilities.invokeLater(new Runnable() {
       public void run() {
         new ListDemo();
       }
    });
   }
}
```





JAVAFX

- Оригиналната GUI рамка на Java е AWT
- Беше последвана от Swing, която предлагаше много по-добър подход
- Въпреки че Swing е много успешен проект, в определени случаи е трудно да се създаде "визуалната искра", която много от днешните приложения изискват
- Освен това концептуалната основа на дизайна на GUI рамки напредна
- За да се справят по-добре с изискванията на съвременния графичен интерфейс и напредъка в дизайна на графичен интерфейс, е необходим нов подход
- Резултатът е JavaFX
  - Java GUI от следващо поколение

JAVAFX

- Дали JavaFX е проектиран като заместител на Swing?
  - Отговорът е: Да
- Въпреки това Swing ще бъде част от Java програмирането за известно време
  - Съществува голямо количество наследен Swing код
  - Освен това, огромен брой програмисти, знаят програмирането със Swing
- Въпреки това JavaFX очевидно е позиционирана като платформа за бъдещето
  - Очаква се през следващите няколко години JavaFX да замени Swing за нови проекти и много Swing базирани приложения ще мигрират към JavaFX
  - Просто казано: JavaFX е нещо, което никой програмист на Java не може да си позволи да пренебрегне

ПРИЛИКИ И РАЗЛИЧИЯ

- Няколко ключови концепции и функции, които трябва да се разберат
- Въпреки че JavaFX има прилики с Swing, съществуват също съществени различия
  - Напр., аналогично на Swing, JavaFX компонентите са леки и събитията се обработват лесно и безпроблемно
- Въпреки това, общата организация на JavaFX и отношението на основните компоненти се различава значително от Swing

JAVAFX ПАКЕТИ

- Рамката JavaFX съдържа пакети, започващи с префикса javafx
- Актуално съществуват от 30 JavaFX пакета в библиотека

STAGE И SCENE

- Централната метафора в JavaFX е *stage*
- Когато една сцена се "играе" тя съдържа *scene*
- Така stage дефинира едно пространство, а scene определя какво се случва в това пространство
  - Т.е. stage е контейнер за scene, а scene е контейнер за елементите, които представляват сцената
- В резултат на това всички JavaFX приложения имат поне един *stage* и една *scene* сцена
- Класовете Stage и Scene

ВЪЗЛИ

- Отделните елементи на сцената се наричат възли (nodes)
  - Напр., един бутон е възел
- Възлите могат също да се състоят от групи възли
- Освен това, един възел може да има възел-наследник
  - Родителски възел или разклонен възел
  - Възлите без наследници са терминални възли и се наричат листа
- Клас Node

ГРАФИ ОТ ВЪЗЛИ

- Колекцията от всички възли в една scene създава това, което се нарича scene graph - който се състои от дърво
- В този граф има един специален тип възел, наречен корен
- Това е възелът на най-високо ниво и е единственият възел в графа, който няма родител
- С изключение на корена, всички други възли имат родители и всички възли пряко или косвено произхождат от корена

ОФОРМЛЕНИЯ

- JavaFX предоставя няколко панела за оформление, които управляват процеса на поставяне на елементи в една сцена
- Напр.,:
  - Класът FlowPane осигурява оформление на потока
  - Класът GridPane поддържа подреждане на базата на редове/колони
  - Класът BorderPane – аналогичен на BorderLayout на AWT
- Тези класове наследяват от Node
- Оформленията са пакетирани в javafx.scene.layout.

БЛАГОДАРЯ ЗА ВНИМАНИЕТО!

КРАЙ “ВЪВЕДЕНИЕ В ООП”

Distributed @learning center