Лекція 9

Класи графічних компонентів

Розглянемо перелік основних графічних компонентів Swing. Всі вони також можуть використовуватися і як контейнери, оскільки успадковуються від класу `java.awt.Container`.

- *JLabel* – мітка, напис;
- *JButton* – кнопка;
- *JTextField* – однорядкове текстове поле;
- *JTextArea* – багаторядкове текстове поле з можливістю форматування (як у форматі HTML, так і в форматі RTF);
- *JTextPane* – багаторядковий текстовий надпис з можливістю форматування;
- *JPasswordField* – текстове поле для скритого введення;
- *JCheckBox* – кнопка-прапорець;
- *JRadioButton* – перемикачі, радіо-кнопки, які часто використовуються з компонентом *ButtonGroup*;
- *JComboBox* – випадний список;
- *JProgressBar* – компонент для наглядного відображення числа в заданому діапазоні;
- *JSlider* – компонент, що дозволяє вибрати значення із заданого діапазону;
- *JSpinner* – компонент, що дозволяє вибрати значення із вказаної послідовності;
- *JList* – список;
- *JTable* – таблиця;
- *JTree* – дерево.

Компоненти поділяють на прості і складні. Прості компоненти не вимагають для своєї роботи ніяких додаткових об'єктів і структур даних. Прикладами таких компонентів є кнопки, написи, однорядкові поля редагування.

Складні компоненти, навпаки, використовуються для відображення великих обсягів структурованих даних. Для внутрішнього представлення даних вони використовують спеціальні об'єкти – *моделі даних*. Сюди відносяться списки, дерева, таблиці, текстові області з форматуванням.

У цій лекції будемо знайомитися з простими компонентами, складні ж будуть розглядатися далі в даному курсі.

Прості компоненти Swing

Клас JComponent

Всі візуальні компоненти бібліотеки Swing успадковані від класу `JComponent`. Сам цей клас є абстрактними і безпосередньо не використовується, але всі візуальні компоненти успадковують його методи. Розглянемо найбільш корисні з них.

`setEnabled(boolean enabled)` використовується для управління активністю компонента. В результаті виклику цього методу з параметром false компонент переходить в неактивний стан. Для кожного спадкоємця `JComponent` «неактивний» стан може бути перевизначений по-різному. Наприклад, неактивна кнопка не натискається, не реагує на мишу і відображається сірим кольором. Метод `isEnabled()` повертає true, якщо елемент активний і false в іншому випадку.

`setVisible(boolean visible)` управляє видимістю компонента. Цей метод вже використовувався для відображення вікна `JFrame`. Більшість елементів управління, на відміну від вікна, за замовчуванням є видимими, тому після їх створення в попередніх прикладах метод `setVisible()` не викликався. Метод `isVisible()` повертає false, якщо елемент невидимий і true в іншому випадку.

Метод `setBackground(Color color)` дозволяє змінити колір фону компонента. Однак ефект спостерігається лише для непрозорих компонентів (деякі компоненти, наприклад мітка `JLabel`, за замовчуванням є прозорими). Прозорістю компонента можна керувати за допомогою метода `setOpaque(boolean opaque)`, в якому параметр true означає непрозорість. Методи `getBackground()` і `isOpaque()` відповідно повертають поточний колір фону і непрозорість компонента.

Мітка JLabel

Мітка – це один з найпростіших компонентів. Зазвичай мітка являє собою звичайний текст, який виводиться в заданому місці вікна і використовується для виведення допоміжної текстової інформації: підписів до інших елементів або інструкцій та попереджень для користувача. У Swing мітка може відображати форматований текст, а також може відображати картинку.

Текст і картинка для мітки зазвичай задаються в її конструкторі. Вона має кілька конструкторів з різними параметрами, зокрема:

`JLabel(String text)` – створює мітку з написом text;

`JLabel(Icon image)` – створює мітку із картинкою image;

`JLabel(String text, Icon image, int align)` – створює мітку з написом text і картинкою image. Третій параметр задає вирівнювання тексту разом із картинкою. Для нього може бути використана одна з констант, описаних в інтерфейсі `SwingConstants`: `LEFT`, `RIGHT`, `CENTER`.

Для прикладу створимо вікно з міткою за допомогою третього конструктора (рис. 6.9). Як і раніше, будемо використовувати два класи, один з яких назвемо `SimpleWindow` і успадкуємо його від класу вікна `JFrame`. У його конструкторі будуть створюватися і розміщуватися всі елементи вікна. Другий клас буде створювати це вікно і відображати його на екрані (його код буде таким же, як і в попередніх прикладах).

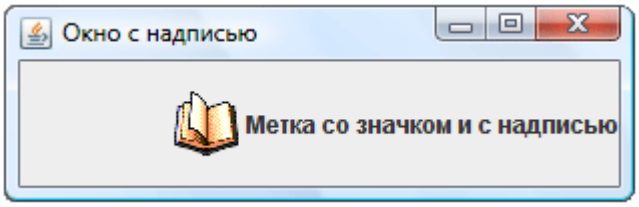

Рис. 6.9. Мітка із картинкою та текстом

Введемо в конструктор класу SimpleWindow наступний код:

```
SimpleWindow() {
    super("Вікно з написом");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    JLabel label = new JLabel("Мітка із картинкою і написом",
                              new ImageIcon("book.png"), SwingConstants.RIGHT);
    getContentPane().add(label);
    pack();
}
```

Щоб переконатися, що вирівнювання по правому краю працює, необхідно трохи розтягнути вікно, щоб ширина мітки стала більше оптимальної.

У бібліотеці Swing всі компоненти можуть відображати форматований текст в форматі HTML. Для цього необхідно, щоб рядок, що встановлюється в якості напису, починався з тега `<html>`. Після цього можна використовувати будь-які теги мови HTML версії 3.2, і вони будуть перетворюватися у відповідні атрибути форматування (рис. 6.10). У цьому легко переконатися, змінивши в попередньому прикладі рядок з викликом конструктора на наступний:

```
JLabel label = new JLabel("<html> До цієї мітки застосовно" +
    "HTML-форматування, включаючи: <ul><li> <i>курсив</i>," +
```

```
"<li><b>напівжирний</b> <li><font size = +2> збільшення розміру </font>" +
"<li>маркований список </ul>");
```

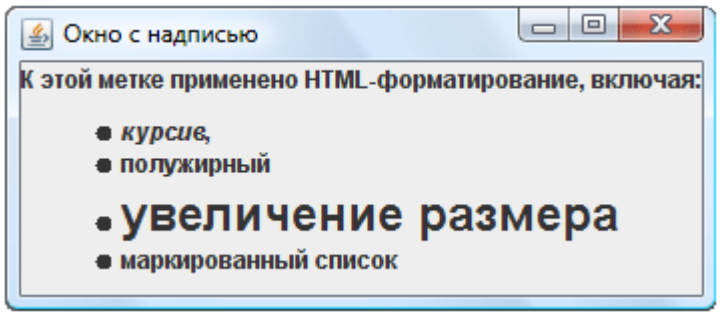

Рис. 6.10. Мітка з форматуванням

Перелічимо найбільш вживані методи класу JLabel:

`getText()` – повертає поточний текст напису мітки;

`setText(String text)` – задає новий текст напису;

`getIcon()` – повертає значок мітки;

`setIcon(Icon image)` – встановлює новий значок. В якості значка зазвичай використовується об'єкт вже знайомого нам простого класу `ImageIcon`.

`getVerticalAlignment()`, `setVerticalAlignment(int align)`, `getHorizontalAlignment()`, `setHorizontalAlignment(int align)` – ці чотири методи дозволяють отримати поточне або встановити нове вирівнювання (по горизонталі і вертикалі) мітки щодо її меж. Можливі положення описані в інтерфейсі `SwingConstants`.

`getVerticalTextPosition()`, `setVerticalTextPosition(int align)`, `getHorizontalTextPosition()`, `setHorizontalTextPosition(int align)` – ці чотири методи дозволяють отримати поточне або встановити нове вирівнювання тексту щодо значка. Можливі положення описані в інтерфейсі `SwingConstants`.

`getIconTextGap()`, `setIconTextGap(int gap)` – дозволяють отримати або задати відстань між текстом і значком мітки в пікселях.

Кнопка JButton

Кнопка являє собою прямокутник з текстом (та/або значком), який можна «натиснути» за допомогою миші або клавіатури, щоб викликати певну дію. Кнопка створюється одним з п'яти конструкторів, зокрема `JButton()`, `JButton(String text)`, `JButton(Icon icon)`, `JButton(String text, Icon icon)`, параметри яких говорять самі за себе.

Крім звичайного значка можна призначити кнопці ще кілька значків для різних станів. Метод `setRolloverIcon(Icon icon)` дозволяє задати значок, який буде з'являтися при наведенні на кнопку миші; `setPressedIcon(Icon icon)` – являє собою значок для кнопки в натиснутому стані і `setDisableIcon(Icon icon)` – це значок для неактивної кнопки. Кожному з цих методів відповідає метод get.

Метод `setMargin(Insets margin)` дозволяє задати величину відступів від тексту напису на кнопці до її полів. Об'єкт класу `Insets`, який передається в цей метод, може бути створений конструктором з чотирма цілочисловими параметрами, які задають величину відступів: `Insets(int top, int left, int bottom, int right)`. Метод `getMargin()` повертає величину поточних відступів у вигляді об'єкта того ж класу.

Всі методи класу `JLabel`, описані раніше, присутні і в класі `JButton`. За допомогою цих методів можна змінювати значок і текст напису на кнопці, а також управляти їх взаємним розташуванням один щодо одного і щодо краю кнопки (з урахуванням відступів).

За допомогою методів

```
setBorderPainted(boolean borderPainted),
```

```
setFocusPainted(boolean focusPainted),
setContentAreaFilled(boolean contentAreaFilled)
```

можна відключати (параметром false) і включати назад (параметром true) промальовування рамки, промальовування фокуса (кнопка, на якій знаходиться фокус, виділяється пунктирним прямокутником) і зафарбовування кнопки в натиснутому стані.

Для прикладу створимо кнопку зі значком і з написом, змінимо її відступи і розташування тексту щодо значка: текст буде вирівняний ліворуч і вгору щодо значка (рис. 6.11).

```
SimpleWindow() {
    super("Вікно з кнопкою");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    JButton button = new JButton("Кнопка", new ImageIcon("book.png"));
    button.setMargin(new Insets(0, 10, 20, 30));
    button.setVerticalTextPosition(SwingConstants.TOP);
    button.setHorizontalTextPosition(SwingConstants.LEFT);
    getContentPane().add(button);
    pack();
}
```

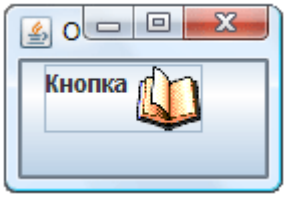

Рис. 6.11. Кнопка зі значком і написом

Компоненти JToggleButton, JCheckBox, JRadioButton

`JToggleButton` представляє собою компонент-кнопку, яка може перебувати в двох станах: натиснутому і відпущеному. Якщо користувач натискає таку кнопку, вона змінює свій стан. Таким чином ведуть себе кнопки форматування на панелі інструментів текстового редактора. Кнопка не тільки встановлює або прибирає курсивне зображення в виділеному тексті, але і сигналізує про його наявність чи відсутність.

Основний конструктор – `JToggleButton(String text, Icon icon, boolean selected)` – створює кнопку із заданими написом, значком і поточним станом. Зауважимо, що кнопку можна перевести в потрібний стан програмним шляхом, викликавши метод `setSelected(boolean selected)`. Метод `isSelected()` повертає true, якщо кнопка вибрана (тобто знаходиться в натиснутому стані) і false у протилежному випадку.

Від класу `JToggleButton` успадкований клас `JCheckBox` – «прапорець». Цей клас має точно такий же набір конструкторів і методів, тобто таку саму поведінку. Єдина відмінність між цими класами проявляється у зовнішньому вигляді.

Схожим чином функціонує клас перемикач або радіо-кнопка `JRadioButton`, що зовні виглядає як кружок. Самі по собі компоненти `JCheckBox` і `JRadioButton` поводяться абсолютно однаково (як і їх спільний предок `JToggleButton`), але ці класи використовують по-різному. Зокрема, `JRadioButton` передбачає вибір єдиної альтернативи з декількох можливих: зазвичай кілька таких об'єктів об'єднуються в одну групу (найчастіше ця група візуально позначається рамкою) і при виборі одного з елементів групи попередній обраний елемент переходить в стан «не вибраний». Для цього використовується спеціальний контейнер `ButtonGroup` – група із взаємовиключенням, яка створюється конструктором без параметрів. Якщо додати в один такий контейнер кілька елементів `JRadioButton`, то обраним завжди буде тільки один з них.

Загалом до `ButtonGroup` можуть бути додані не тільки перемикачі, але і прапорці, і кнопки вибору, і навіть звичайні кнопки `JButton`. Але згідно загально прийнятому підходу у групу із

взаємовиключенням слід об'єднувати об'єкти `JRadioButton`, а в деяких випадках `JToggleButton`, але не інші види кнопок.

Метод `add(AbstractButton button)` додає елемент в групу. Метод `getElements()` повертає всі її елементи у вигляді колекції `Enumeration`. Далі можна перебрати елементи такої колекції ітератором і знайти вибраний користувачем елемент.

Розглянемо приклад, в якому створимо дві кнопки вибору, два прапорці і два перемикачі (рис. 6.12). Кнопки вибору і перемикачі об'єднані в групи `ButtonGroup`. Для того, щоб обвести кожну пару елементів рамкою, необхідно розташувати цю пару елементів на окремій панелі.

```
SimpleWindow() {
    super("Приклад з кнопками вибору, прапорцями і перемикачами");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    ImageIcon icon = new ImageIcon("book.png");
    Box mainBox = Box.createVerticalBox();
    Box box1 = Box.createVerticalBox();
    JToggleButton tButton1 = new JToggleButton("Кнопка вибору 1");
    JToggleButton tButton2 = new JToggleButton("Кнопка вибору 2", icon);
    ButtonGroup bg = new ButtonGroup();   // створюємо групу взаємного виключення
    bg.add(tButton1); bg.add(tButton2);   // додаємо кнопки до групи
    box1.add(tButton1); box1.add(tButton2);   // додаємо їх також до панелі box1
    box1.setBorder(new TitledBorder("Кнопки вибору"));
    Box box2 = Box.createVerticalBox();
    JCheckBox check1 = new JCheckBox("Прапорець 1");
    JCheckBox check2 = new JCheckBox("Прапорець 2", icon); box2.add(check1);
    box2.add(check2);   // додаємо флажки на панель box2
    box2.setBorder(new TitledBorder("Прапорці"));
    Box box3 = Box.createVerticalBox();
    JRadioButton rButton1 = new JRadioButton("Перемикач 1");
    JRadioButton rButton2 = new JRadioButton("Перемикач 2", icon);
    bg = new ButtonGroup();   // створюємо групу взаємного виключення
    bg.add(rButton1); bg.add(rButton2);   // робимо радіо-кнопки взаємновиключаючими
    box3.add(rButton1); box3.add(rButton2);   // додаємо радіокнопки на панель box3
    box3.setBorder(new TitledBorder("Перемикачі"));
    mainBox.add(box1);
    mainBox.add(box2);
    mainBox.add(box3);
    setContentPane(mainBox);
    pack();
}
```

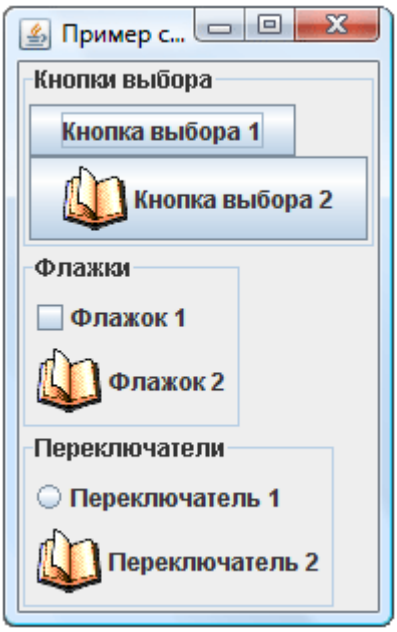

Рис. 6.12. Демонстрація різних видів кнопок

На цьому прикладі ми бачимо, що у прапорців і перемикачів значок замінює індикатор виділення. Однак значок не вказує стан, тому необхідно встановити другий значок для стану "selected" за допомогою метода `setSelectedIcon (Icon icon)`:

```
check2.setSelectedIcon(new ImageIcon("2.png"));
rButton2.setSelectedIcon(new ImageIcon("2.png"));
```

Текстове поле JTextField

Це простий і часто використовуваний компонент, призначений для введення невеликих за обсягом текстових даних в один рядок. Текстове поле створюється одним з конструкторів:

`JTextField(int columns)` – створює порожнє текстове поле, ширина якого достатня для розміщення `columns` символів. Користувач може вводити в текстове поле рядок будь-якої довжини: вона просто буде прокручуватися.

`JTextField(String text)` – текстове поле створюється з початковим текстом text.

`JTextField(String text, int columns)` – дозволяє встановити як ширину, так і початковий текст.

Текст заноситься в поле методом `setText(String text)`. Метод `getText()` повертає вміст текстового поля цілком, а `getText(int offset, int length)` – лише фрагмент тексту.

Підтримується виділення частини тексту як програмним шляхом, так і в результаті дій користувача. При цьому метод `getSelectedText()` дає можливість отримати виділену частину тексту. Заміна виділеного тексту іншим здійснюється за допомогою методу `replaceSelection(String content)`. Методи `getSelectionStart()` і `getSelectionEnd()` повертають межі виділеної ділянки, а методи `setSelectionStart(int start)` і `setSelectionEnd(int end)` змінюють їх.

Метод `getCaretPosition()` повертає позицію курсора (каретки) в текстовому полі, в той час як метод `setCaretPosition(int position)` дозволяє задати її програмно. Метод `setCaretColor(Color color)` дозволяє змінити колір курсора.

Текст в полі за замовчанням вирівнюється за лівим краєм. Змінити це можна методом `setHorizontalAlignment(int align)`, якому в якості параметра передається одна з констант вирівнювання, визначених у цьому ж класі `JTextField`: `LEFT`, `CENTER`, `RIGHT`.

Поле для введення пароля JPasswordField

`JPasswordField` є прямим нащадком `JTextField`, тому для нього справедливо все сказане вище. Відмінність полягає в тому, що весь введений в нього текст прихований: він замінюється зірочками або іншим символом, встановити який дозволяє метод `setEchoChar(char echo)`, а отримати – `getEchoChar()`.

`JPasswordField` застосовується для введення пароля. Метод `getText()` дозволяє отримати введений пароль у вигляді рядка, але користуватися ним не рекомендується, оскільки при бажанні стороння особа може проаналізувати вміст оперативної пам'яті і перехопити пароль. Тому треба в цьому випадку використовувати метод `getPassword()`, який повертає посилання на масив символів `char[]`.

Після того, як введений пароль буде оброблено (наприклад, порівняно з правильним паролем) рекомендується заповнити цей масив нулями, щоб слідів в оперативній пам'яті не залишилося.

Область для введення тексту JTextArea

`JTextArea` також є нащадком `JTextField` і успадковує всі його методи. На відміну від текстового поля, область для введення тексту дозволяє ввести кілька рядків тексту. У зв'язку з цим `JTextArea` пропонує ряд додаткових функцій.

По-перше, можна налаштувати автоматичне перенесення довгих рядків тексту за допомогою методу `setLineWrap(boolean lineWrap)`. Якщо передати параметр true, то довгі рядки будуть переноситися, а не виходити за межі компонента. По-друге, можна налаштувати перенесення слів

за допомогою методу `setWrapStyleWord(boolean wrapStyle)`. Якщо викликати цей метод з параметром true, то слова не будуть розриватися при перенесенні.

Для створення `JTextArea` найчастіше використовують конструктор `JTextArea(int rows, int columns)`, що встановлює видимі висоту (кількість рядків) і ширину (кількість символів) компонента.

Для роботи зі своїм вмістом `JTextArea` додатково пропонує два зручних методи. Це метод `append(String text)`, який додає рядок text в кінець вже наявного тексту та метод `insert(String text, int position)`, що вставляє його в позицію position.

Покажемо роботу розглянутих текстових компонентів на наочному прикладі. Створимо просте вікно, в якому розмістимо три різні текстові поля за допомогою менеджера `BorderLayout` (рис. 6.13).

```
SimpleWindow() {
    super("Приклад текстових компонентів");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    JTextField textField = new JTextField("Текстове поле", 20);
    textField.setCaretColor(Color.RED);
    textField.setHorizontalAlignment(JTextField.RIGHT);
    JPasswordField passwordField = new JPasswordField(20);
    passwordField.setEchoChar('$');
    passwordField.setText("пароль");
    JTextArea textArea = new JTextArea(5, 20);
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
    for (int i = 0; i <= 20; i++) textArea.append("Текст");
    getContentPane().add(textField, BorderLayout.NORTH);
    getContentPane().add(textArea);
    getContentPane().add(passwordField, BorderLayout.SOUTH);
    pack();
}
```

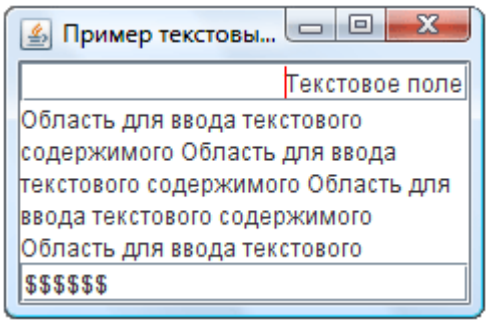

Рис. 6.13. Демонстрація різних компонентів для введення тексту

Замініть по черзі true на false у викликах методів `setLineWrap()` і `setWrapStyleWord()`. При цьому спостерігаються зміни роботи компонента. Також змінюючи розміри вікна можна бачити яким чином текст перебудовується під доступний йому простір.

Панель прокрутки JScrollPane

Спостерігаючи за поведінкою компонента `JTextArea` в попередньому прикладі, легко можна виявити проблеми, які виникають, коли текст не вміщується в поле. Залежно від використовуваного менеджера розміщення текст або обрізається, йдучи за межі компоненту, або розсовує ці межі. В обох випадках текст залишається обмежений розміром вікна. У таких випадках як правило використовують смуги прокрутки.

Додати для компонента смуги прокрутки доволі легко. Для цього служить компонент `JscrollPane` – панель прокрутки. Вона створюється поверх компонента, який має прокручуватися. Наприклад, аби текстова область `textArea` з попереднього прикладу мала полоси прокрутки, необхідно замінити рядок:

```
getContentPane().add(textArea);
```

на наступний:

```
getContentPane().add(new JScrollPane(textArea));
```

Тут створюється панель з смугами прокрутки і в неї поміщається об'єкт `textArea`, а сама панель додається в панель вмісту вікна. Тепер текст вільно прокручується. В разі ж застосування менеджера `FlowLayout` або `BoxLayout` компонент `JTextArea` не буде підлаштовуватися під свій вміст, а матиме бажаний розмір, відповідний параметрам конструктора, і відображати смуги прокрутки за необхідності.

`JScrollPane` дозволяє задавати правила відображення смуг прокрутки за допомогою двох властивостей `horizontalScrollBarPolicy` і `verticalScrollBarPolicy`. Вони мають наступні можливі значення:

- HORIZONTAL_SCROLLBAR_ALWAYS, VERTICAL_SCROLLBAR_ALWAYS – завжди відображати смуги прокрутки, незалежно від розміру вмісту;
- HORIZONTAL_SCROLLBAR_AS_NEEDED, VERTICAL_SCROLLBAR_AS_NEEDED – відображати смуги прокрутки, тільки коли вміст не вміщується всередині панелі;
- HORIZONTAL_SCROLLBAR_NEVER, VERTICAL_SCROLLBAR_NEVER – не відображати смуги прокрутки.

Всі ці константи визначені в інтерфейсі `ScrollPaneConstants`.

Панель інструментів JToolBar

Значна більшість програмних продуктів надають зручні інструментальні панелі, розташовані уздовж меж вікна програми і містять кнопки, списки, що випадають та інші елементи управління, які зазвичай відповідають командам меню. У Swing для інструментальних панелей розроблений багатофункціональний компонент `JToolBar`.

Для прикладу створимо вікно з менеджером розташування `BorderLayout` та розмістимо по центру область для введення тексту `JTextArea`, а до верхнього краю прикріпимо інструментальну панель з трьома кнопками і одним роздільником (рис. 6.14):

```
SimpleWindow() {
    super("Приклад використання JToolBar");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    JTextArea textArea = new JTextArea(5, 20);
    getContentPane().add(textArea);
    JToolBar toolBar = new JToolBar("Панель інструментів");
    toolBar.add(new JButton("Кнопка 1"));
    toolBar.add(new JButton("Кнопка 2"));
    toolBar.addSeparator();
    toolBar.add(new JButton("Кнопка 3"));
    getContentPane().add(toolBar, BorderLayout.NORTH);
    pack();
}
```

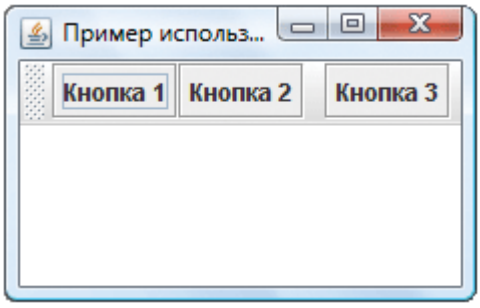

Рис. 6.14. Демонстрація панелі інструментів

Запустимо цей приклад і поекспериментуємо з інструментальною панеллю. Наприклад, спробуємо від'єднати її від верхнього краю вікна і прикріпити до будь-якого іншого краю. Потім від'єднаємо її від меж вікна так, щоб панель стала самостійним вікном. При цьому панель завжди відображається над батьківським вікном, навіть якщо воно, а не панель, є активним. Якщо закрити самостійну панель кнопкою з хрестиком, вона повернеться в своє вікно, тобто в те місце, де вона була закріплена останній раз.

Цей простий приклад продемонстрував базові можливості інструментальної панелі. Тепер наведемо деякі корисні методи `JToolBar`:

Конструктор `JToolBar(String title)` створює панель із заданим заголовком. За замовчанням створюється горизонтальна панель, яка призначена для прикріплення до верхньої або нижньої межі батьківського контейнера та має розміщення BorderLayout. Для створення вертикальної панелі використовується конструктор `JToolBar(String title, int orientation)`, де параметр orientation задається константою SwingConstants.VERTICAL. Інші конструктори JToolBar() і `JToolBar (int orientation)` створюють панель без заголовка.

`setFloatable(boolean floatable)` – дозволяє або забороняє користувачу відкріплювати панель від місця її початкового розташування. Відповідно метод `isFloatable()` повертає true, якщо відкріплювати панель дозволено.

`add(Component component)` – додає на інструментальну панель новий елемент управління. Взаємопов'язані групи елементів управління прийнято розділяти за допомогою лінії-роздільника: `addSeparator()`.

Випадний список JComboBox

Такий список містить кілька варіантів, з яких користувач може вибрати один і тільки один, або ввести свій власний, якщо випадний список це дозволяє.

Випадний список можна створити конструктором за замовчуванням JComboBox(), після чого додавати в нього елементи методом addItem(Object item), що додає новий елемент в кінець списку, або методом insertItemAt(Object item, int index), який вставляє елемент в задану позицію. Зазвичай найпростіше в конструкторі вказати відразу всі елементи списку. Таких конструкторів два: JComboBox(Object[] elements) і JComboBox(Vector elements). Ці конструктори працюють однаково, використовуючи різне подання елементів.

Найчастіше у випадний список додають елементи-рядки, проте, як це випливає з сигнатур описаних вище методів, він *може містити будь-які об'єкти*. Будь-який об'єкт перетворюється в рядок методом toString(), і саме цей рядок представлятиме його у випадному списку.

Метод getItemAt(int index) дозволяє звернутися до елементу списку за номером.

Метод removeAllItems() видаляє з JComboBox всі елементи, а метод removeItem(Object item) видаляє лише один вказаний елемент.

Метод getSelectedIndex() дозволяє отримати індекс обраного користувачем елемента (елементи нумеруються, починаючи з нуля), а метод getSelectedItem() повертає сам обраний об'єкт. Змінити вибір можна і програмно, скориставшись методом setSelectedIndex(int index) або setSelectedItem(Object item).

Для того щоб користувач міг ввести свій варіант, який відсутній в списку, потрібно встановити властивість editable: setEditable(true).

Тепер розглянемо приклад, в якому створюється випадний список з трьох елементів і вибирається другий елемент (рис. 6.15). Рядок, що представляє третій елемент, містить маркування HTML.

```
SimpleWindow() {
    super("Приклад використання JComboBox");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    String[] elements = new String[]
        {"Вася", "Петя", "<html><font size = +1 color = yellow>Іван</font>"};
    JComboBox combo = new JComboBox(elements);
    combo.setSelectedIndex(1);
    JPanel panel = new JPanel();
    panel.add(combo);
```

```
    setContentPane(panel);
    setSize(200, 200);
}
```

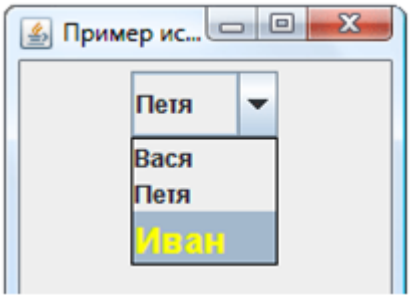

Рис. 6.15. Демонстрація випадного списку

Повзунок JSlider

Повзунок дозволяє користувачеві вибрати певну кількість з діапазону доступних значень, наочно представивши цей діапазон (рис. 6.16). Проте повзунок має один недолік – він займає досить багато місця.

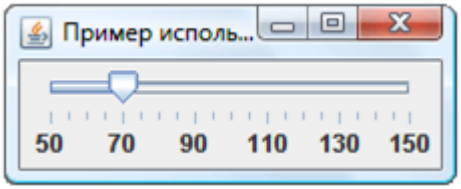

Рис. 6.16. Демонстрація повзунка

Основний конструктор повзунка: JSlider(int orientation, int min, int max, int value), де перший параметр – орієнтація повзунка (HORIZONTAL або VERTICAL), інші параметри вказують відповідно мінімальне, максимальне і поточне значення. Змінити їх дозволяють методи setOrientation(int), setMinimum(int min), setMaximum(int max), setValue(int value), а отримати поточні – відповідні їм методи get. Найчастіше використовується метод getValue() – аби визначити, яке значення вибрав користувач.

Шкала повзунка може мати ділення. Метод setMajorTickSpacing(int spacing) задає ціну великої поділки шкали, а метод setMinorTickSpacing(int spacing) – відповідно ціну малої поділки. В свою чергу, метод setPaintTicks(boolean paint) включає або відключає виведення цих поділок. Метод setSnapToTicks(boolean snap) включає або відключає «прилипання» повзунка до поділок: якщо викликати цей метод з параметром true, користувач зможе вибрати за допомогою повзунка тільки такі значення, що відповідають поділкам. Нарешті, метод setPaintLabels(boolean paint) включає або відключає виведення написів під великими поділками.

Приклад використання перерахованих методів:

```
SimpleWindow() {
    super("Приклад використання JSlider");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    JSlider slider = new JSlider(JSlider.HORIZONTAL, 50, 150, 70);
    slider.setMajorTickSpacing(20);
    slider.setMinorTickSpacing(5);
    slider.setPaintTicks(true);
    slider.setPaintLabels(true);
    slider.setSnapToTicks(true);
    JPanel panel = new JPanel();
    panel.add(slider);
    setContentPane(panel);
    pack();
}
```

Панель із вкладками JTabbedPane

Створити панель з вкладками можна простим конструктором, в якому визначається тільки місце розташування ярликів (LEFT, RIGHT, TOP або BOTTOM). Проте іноді буває корисний конструктор

```
JTabbedPane(int orientation, int layout);,
```

де другий параметр приймає значення, що відповідають константам:

- SCROLL_TAB_LAYOUT (якщо всі ярлики не вміщуються, то появляється смуга прокрутки);
- WRAP_TAB_LAYOUT (ярлики можуть розташовуватись в кількох рядах).

Після цього можна додавати вкладки методом addTab(), що має кілька варіантів. Зокрема, метод addTab(String title, Component tab) додає закладку із зазначенням тексту ярлика, а метод addTab(String title, Icon icon, Component tab) дозволяє задати також і значок до ярлика. В якості вкладки зазвичай служить панель з розміщеними на ній елементами управління.

Тепер створимо панель з десятьма вкладками, на кожній з яких помістимо кнопку (рис. 6.17). Всі ці вкладки створимо в циклі.

```
SimpleWindow() {
    super("Приклад використання JTabbedPane");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP,
                                             JTabbedPane.WRAP_TAB_LAYOUT);
    for (int i = 1; i <= 10; i++) {
        JPanel panel = new JPanel();
        panel.add(new JButton("Кнопка № " + i));
        tabbedPane.add("Панель " + i, panel);
    }
    getContentPane().add(tabbedPane);
    setSize(300, 200);
}
```

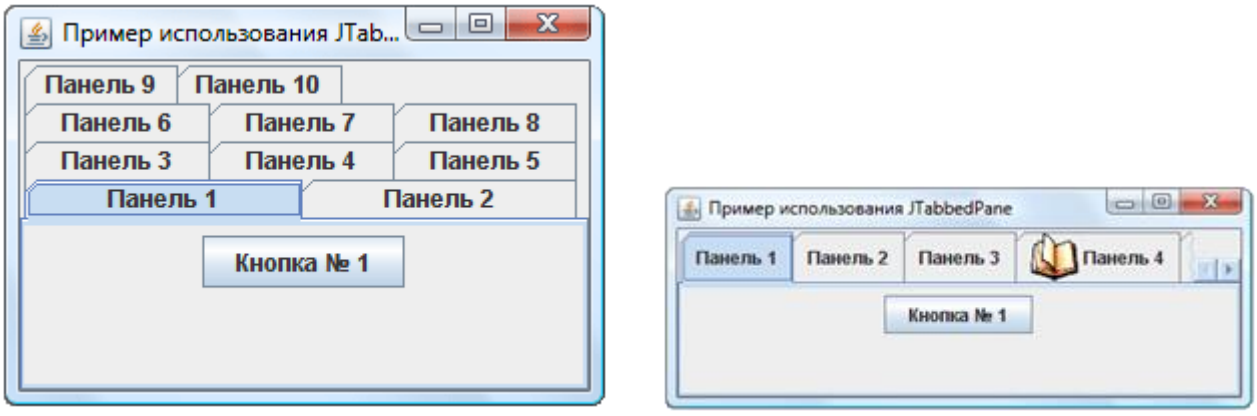

Рис. 6.17. Панель з різними варіантами розміщення вкладок

Список JList

Список JList – це один із важких компонентів, для ефективної роботи з якими необхідно розуміння концепції «Модель-Вид». Компоненти JTree (дерево) і JTable (таблиця) ще складніші, і їх необхідно розглядати окремо. Що стосується списку, то частина його можливостей може бути використана без заглиблення в деталі патерну «Модель-Вид».

Список містить групу елементів, аналогічно випадному списку JComboBox, але, по-перше, на екрані видно одночасно кілька елементів списку і, по-друге, користувач може вибрати в списку не один елемент, а кілька (за умови, якщо встановлений відповідний режим виділення).

Створити список можна на основі масиву Object[] або вектору Vector елементів. Метод setVisibleRowCount(int count) встановлює кількість видимих елементів списку. Інші пункти будуть виходити за його межі (або прокручуватися, якщо помістити список в JScrollPanel).

JList може зберігати не тільки рядки, але будь-які об'єкти. Для відображення цих об'єктів використовується метод toString().

За замовчуванням користувач може вибрати в списку будь-яке число елементів, утримуючи клавішу Ctrl. Режим виділення можна змінити методом setSelectionMode(int mode), де параметр задається однією з констант класу ListSelectionModel:

- SINGLE_SELECTION – можна виділити тільки один елемент,
- SINGLE_INTERVAL_SELECTION – можна виділити кілька елементів, розташованих підряд,
- MULTIPLE_INTERVAL_SELECTION – можна виділити довільну кількість елементів, як суміжних так і не суміжних.

Один виділений елемент списку можна отримати методом getSelectedValue(). Якщо виділено кілька елементів, тоді метод поверне перший з них. Метод getSelectedValues() повертає всі виділені елементи у вигляді масиву Object[]. Аналогічно працюють методи getSelectedIndex() і getSelectedIndices(), але повертають вони не самі виділені елементи, а їх індекси. До всіх цих методів є також парні методи set, так що змінювати виділення елементів списку можна і програмно.

Наступний приклад ілюструє деякі з можливостей JList (рис. 6.18):

```
SimpleWindow() {
    super("Приклад з JList");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    Object[] elements = new Object[]{ "Ковбаса",
              "<html><font color = red>Масло", "Згущене молоко"};
    JList list = new JList(elements);
    list.setVisibleRowCount(5);
    list.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
    list.setSelectedIndices(new int[]{1, 2});
    getContentPane().setLayout(new FlowLayout());
    getContentPane().add(new JScrollPane(list));
    setSize(200, 150);
}
```

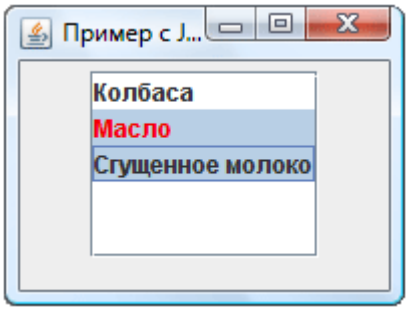

Рис. 6.18. Демонстрація списку

Для того, щоб ефективно додавати і видаляти елементи зі списку, керувати їх відображенням, додавати і видаляти виділені елементи поодинці, необхідно познайомитися з *моделлю даних списку*, що виходить за рамки цієї лекції.

Властивості компонентів

Всі властивості компонентів, що доступні в GUI-редакторі через панель властивостей, можна отримувати і змінювати програмно за допомогою методів set і get. Основні властивості, застосовні до більшості компонентів, такі:

- text – текстовий надпис;

- background – колір заповнення фону;
- foreground – колір тексту;
- font – шрифт тексту, об’єкт класу Font;
- visible – видимість;
- enabled – активність.

Створимо, наприклад, кнопку з жирним червоним написом. Означене можна зробити або за допомогою маркування HTML:

```
JButton myButton = new JButton(
     "<html><b><font color=\"red\">OK</font></b></html>");
```

або використовуючи властивості компонента:

```
JButton b = new JButton("OK”);
b.setForeground( Color.RED );
b.setFont(new Font("Tahoma", Font.BOLD, 14));
```