Лекція 10

Події компонентів Swing

Графічний інтерфейс користувача (GUI) визначає не тільки розташування необхідних елементів управління, але і поведінку додатку у відповідь на дії користувача. Значна частина дій у віконних програмах виконується у відповідь на вибір користувачем команд меню, натискання кнопок, а іноді навіть просто у відповідь на введення нового символу в текстовому полі.

Отже, при створенні додатку необхідно:

1. Виявити події, у відповідь на які потрібна реакція програми.
2. Написати код, який реалізує цю реакцію, тобто обробник подій.
3. Пов’язати обробник події з відповідною подією.

Перший пункт залежить від логіки роботи додатку і перебуває повністю під контролем програміста. ***Обробник*** події – це, як правило, звичайний метод в одному з класів додатку. Обробник будь-якої дії для компонента можна призначити як в GUI-редакторі, так і вручну в коді програми.

Патерн проектування «спостерігач»

У різних мовах програмування і візуальних бібліотеках використовуються різні способи прив'язки процедури (методу), яка повинна викликатися у відповідь на деяку подію, до самої цієї події. У бібліотеці Swing для прив'язки обробника події до компонента, що його викликає, використовується патерн проектування «спостерігач».

Патерни проектування – це стандартні прийоми об'єктно-орієнтованого програмування, що дозволяють оптимальним чином впоратися з нетривіальними, але поширеними задачами в програмуванні. Патерн проектування описує деяке стандартне рішення задачі у вигляді набору класів з чітко визначеними ролями і правилами взаємодії між ними.

Наприклад, патерн «*ітератор*» вирішує задачу перебору елементів довільної колекції, незалежно від її природи. У цьому патерні беруть участь один інтерфейс Iterable і два класи: перший представляє набір елементів і реалізує інтерфейс Iterable, а другий – власне ітератор – реалізує логіку проходу по цьому набору. Даний патерн дозволяє перебрати елементи колекції за допомогою двох простих методів, незалежно від того в якому порядку і за якими правилами відбувається перебір, а також від того, де фізично зберігаються перебирувані елементи.

Патерн проектування «***спостерігач***» застосовується, коли один об'єкт повинен сповіщати інші про зміни, що з ним відбулися. При цьому сам такий об'єкт називається спостережуваним, а об'єкти, які слід оповістити – ***спостерігачами***.

Подібна взаємодія буде можлива, якщо об'єкт-спостерігач має метод (або кілька методів) із заздалегідь визначеною сигнатурою (тобто ім'ям і параметрами), в якому запрограмована реакція на подію. Тоді в момент очікуваної події спостережуваний об'єкт може викликати відповідний метод свого спостерігача, «сповістивши» тим самим його про подію.

Для того, щоб спостережуваний об'єкт міг викликати метод спостерігача, він повинен мати посилання на об'єкт-спостерігач. Це посилання спостережуваного об'єкту потрібно попередньо передати, і це робиться за допомогою спеціального методу. Таким чином, спостережуваний об'єкт повинен зберігати список посилань на всіх своїх спостерігачів, а також метод для реєстрації нових спостерігачів.

Зауважте, що в даній схемі один спостерігач може бути зареєстрований у декількох спостережуваних об'єктів – тоді він буде однаково реагувати на зміни в кожному з них. З іншого боку, у одного спостережуваного об'єкта може бути кілька спостерігачів – тоді всі вони сповіщаються при виникненні події і виконується кілька незалежних методів-обробників. Це неабияк підвищує гнучкість програмування подій.

Механізм обробки подій Swing

У Swing спостережуваними об’єктами є всі без виключення компоненти інтерфейсу: саме вікно програми, кнопки, меню, поля введення і т. д. Вони можуть повідомити своїм спостерігачам

про певні події, як елементарні (наведення миші, натискання клавіші на клавіатурі), так і високого рівня (зміна тексту в текстовому полі, вибір нового елемента у випадному списку і т. д.).

Спостерігачами виступають об'єкти класів, що реалізують спеціальні інтерфейси Listener. Такі класи в термінології Swing називаються ***слухачами (listeners)***.

Інтерфейс MouseListener

Розглянемо технологію написання слухачів на прикладі слухачів подій миші.

Події миші – це один з найбільш затребуваних видів подій. Будь-який компонент інтерфейсу здатний повідомити про те, що на нього навели мишу чи натиснули по ньому і т. д. Про це будуть сповіщені всі зареєстровані слухачі подій миші.

Слухач подій миші повинен реалізувати інтерфейс **MouseListener**. У цьому інтерфейсі перераховані наступні методи:

- public void mouseClicked(MouseEvent event) – натискання мишею на об’єкті;
- public void mouseEntered(MouseEvent event) – курсор миші зайшов у область об’єкта;
- public void mouseExited(MouseEvent event) – курсор миші вийшов із області об’єкта;
- public void mousePressed(MouseEvent event) – кнопка миші натиснута, коли курсор знаходиться над об’єктом;
- public void mouseReleased(MouseEvent event) – кнопка миші відпущена, коли курсор знаходиться над об’єктом.

Наприклад, нам необхідно описати реакцію на подію натискання мишею кнопки «Увійти до системи». Спостережуваним об’єктом в цьому разі є кнопка стандартного класу JButton, а от клас «слухача» нам необхідно описати самим, реалізувавши інтерфейс MouseListener і помістивши код для обробки події в метод mouseClicked().

Опишемо клас слухача в межах класу вікна SimpleWindow. Обробник події перевірятиме, чи ввів користувач логін «Іван», а потім виведе повідомлення про успішний чи безуспішний вхід в систему:

```
class SimpleWindow extends JFrame {

     private JTextField loginField;
     private JTextField passwordField;

     class ButttonMouseListener implements MouseListener {

         public void mouseClicked(MouseEvent event) {
             if (loginField.getText().equals("Іван")) {
                 JOptionPane.showMessageDialog(null, "Вхід виконано");
             }
             else {
                 JOptionPane.showMessageDialog(null, "Вхід НЕ виконано!");
             }
         }

         public void mouseEntered(MouseEvent event) { }
         public void mouseExited(MouseEvent event) { }
         public void mousePressed(MouseEvent event) { }
         public void mouseReleased(MouseEvent event) { }
     }
     // ...
}
```

В нашій програмі слухач являється *вкладеним* класом всередині класу вікна SimpleWindow щоб він міг легко отримати доступ до змінних loginField і passwordField. Зверніть увагу, що нам довелося перевизначити всі п'ять абстрактних методів MouseListener, хоча тільки один із них реально використовується, а чотири мають порожню реалізацію. Якщо в класі реалізовані не всі абстрактні методи, то неможливо створити об'єкт цього класу.

Далі необхідно створити об'єкт-слухач і пов’язати його із кнопкою методом **addMouseListener(MouseListener listener)**. Це доречно зробити в конструкторі вікна SimpleWindow(), одразу після створення кнопки:

```
JButton ok = new JButton("OK");
ok.addMouseListener(new ButttonMouseListener());
```

Створення слухачів за допомогою анонімних класів

У попередньому прикладі для створення слухача для кнопки, ми створили новий вкладений клас, який виглядає досить громіздко. Якщо, як в нашому випадку, клас описується тільки для одноразового використання, то код можна скоротити, зробивши клас ***анонімним***.

Анонімний клас не має імені, а його оголошення поєднане зі створенням об'єкта цього класу. Всього в програмі може бути створений тільки один об'єкт анонімного класу.

У нашому прикладі необхідний тільки один об'єкт класу ButttonMouseListener. І в інших програмах дуже часто слухач пишеться для обробки подій одного єдиного об'єкта, а значить, використовується в програмі лише один раз: під час прив'язки до цього об'єкта.

Замінимо вкладений клас ButttonMouseListener анонімним. Для цього його старий опис потрібно видалити, а рядок:

```
ok.addMouseListener(new ButttonMouseListener());
```

замінити на наступний код:

```
ok.addMouseListener(
    new MouseListener() {
         public void mouseClicked(MouseEvent event) {
             if (loginField.getText().equals("Іван")) {
                 JOptionPane.showMessageDialog(null, "Вхід виконано");
             }
             else {
                 JOptionPane.showMessageDialog(null, " Вхід НЕ виконано!");
             }
         }
         public void mouseEntered(MouseEvent event) {}
         public void mouseExited(MouseEvent event) {}
         public void mousePressed(MouseEvent event) {}
         public void mouseReleased(MouseEvent event) {}
});
```

Новий варіант коротше, але в ньому настройка графічних компонентів (кнопки) і логіка обробки подій суміщені в одному методі, що неправильно з точки зору ООП: ці дії повинні бути розділені. Тому зловживання анонімними класами може зробити програму непрозорою і нечитабельною. На практиці їх застосовують в разі простих (в кілька рядків) обробників подій. Якщо ж в обробниках знаходиться багато коду, то їх необхідно виносити в окремі класи, які називаються *контролерами*.

Відзначимо, що можна реалізувати інтерфейс слухача в будь-якому класі, додавши до його опису `implements MouseListener` і визначивши в ньому необхідні методи. Зокрема, це можна зробити в самому класі вікна:

```
public class SimpleWindow implements MouseListener {...}
```

Але такий варіант використовується нечасто. Зазвичай в межах одного вікна кілька компонентів мають однотипні події (наприклад, кілька кнопок які можна натискати мишею), а значить необхідно мати кілька різних класів-слухачів, успадкованих від MouseListener.

Клас MouseAdapter

Додавання слухача в нашій програмі виглядає громіздко переважно через те, що крім корисного для нас методу mouseClicked() нам довелося визначати порожні реалізації всіх інших методів MouseListener. Цього можна уникнути.

Клас **MouseAdapter** реалізує інтерфейс MouseListener, визначаючи порожні реалізації для кожного з його методів. Можна успадкувати свій клас слухача від нього і перевизначити тільки ті методи, які нам потрібні.

```
ok.addMouseListener(new MouseAdapter() {
    public void mouseClicked (MouseEvent event){
        if (loginField.getText().equals("Іван")) {
              JOptionPane.showMessageDialog(null, "Вхід виконано");
        }
        else { JOptionPane.showMessageDialog(null, "Вхід НЕ виконано!"); }
    }
});
```

Подібні класи «адаптерів» є для всіх інтерфейсів слухачів Swing, які містять більше одного методу.

Загальна структура слухачів

Візуальні компоненти Swing підтримують цілий ряд різних видів слухачів. При цьому різні компоненти можуть мати різний набір застосовних до них слухачів.

Кожен слухач повинен реалізовувати інтерфейс XxxListener, де Xxx – тип слухача. Практично кожному з цих інтерфейсів (за винятком тих, в яких всього один метод) відповідає порожній клас-заглушка XxxAdapter. Кожен метод інтерфейсу слухача приймає один параметр типу XxxEvent, в якому зібрана вся інформація, що відноситься до події. Ми не будемо детально розглядати методи отримання цієї інформації. Як правилом, їх небагато і їх призначення легко зрозуміти, а перелік методів дасть контекстна підказка IDE.

Щоб прив'язати слухача до компоненту у всіх випадках використовується метод:

```
addXxxListener(XxxListener listener);
```

Наприклад, слухач подій, пов'язаних з коліщатком миші, повинен реалізувати інтерфейс MouseWheelListener, і є клас-заглушка MouseAdapter. В даному випадку один клас-заглушка відповідає відразу декільком інтерфейсам слухачів, пов'язаних з мишею. Методи інтерфейсу MouseWheelListener мають параметр типу MouseWheelEvent, а реєструється слухач методом addMouseWheelListener (MouseWheelListener listener). Ми не будемо повторювати цю інформацію для інших типів слухачів, вони працюють аналогічно. Замість цього ми розглянемо прийоми їх використання.

Універсальний слухач ActionListener

Багато елементів управління (але не всі) мають одну основну подію, яка обробляється значно частіше, ніж інші. Наприклад, для кнопки це натискання, а для списку – зміна обраного елемента. Для відстеження та обробки такої події використовують слухач **ActionListener**, який має один метод:

```
public void actionPerformed (ActionEvent event);
```

ActionListener – це найбільш часто використовуваний слухач. Зазвичай більшість компонентів форми не мають інших, більш спеціалізованих слухачів. Розглянемо кілька типових прикладів.

Приклад 1. Перепишемо обробник для кнопки «Ok» з попереднього прикладу за допомогою ActionListener:

```
ok.addActionListener(
    new ActionListener() {
         public void actionPerformed(ActionEvent event) {
              if (loginField.getText().equals("Іван")) {
                   JOptionPane.showMessageDialog(null, "Вхід виконано");
              } else {
                   JOptionPane.showMessageDialog(null, "Вхід НЕ виконано!");
              }
         }
});
```

Зверніть увагу, що цей обробник не еквівалентний MouseAdapter, оскільки кнопка натискається не тільки мишею, але також і за допомогою клавіші Enter.

Приклад 2. Для текстового поля додамо перевірку введеного значення на допустимість. Нехай необхідно вводити дійсне число, а при неправильному введенні – виділити введений текст червоним кольором:

```
     txt = new JTextField();
     txt.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent arg0) {
               if( !txt.getText().matches("\\d*\\.?\\d+") ) {
                    txt.setForeground( new Color( 255, 100, 100 ) );
               } else {
                    textField.setForeground( Color.BLACK );
               }
          }
     });
     form.getContentPane().add(textField);
}
```

Зверніть увагу, що для текстового поля ActionListener спрацює тільки в разі завершення введення клавішею Enter. Для ситуації, коли користувач перейшов на наступне поле за допомогою миші або клавіші Tab, потрібно обробляти інші події, які ми будемо обговорювати далі.

Приклад 3. Створення вікна з кнопками OK і Cancel

```
import java.awt.*;
import javax.Swing.*;

// Клас додатку
public class App {
     private JFrame form;
     public static void main(String[] args) {
          // Створення вікна
          App app = new App();
          app.form.setVisible(true);
}
     // Ініціалізація вікна
     public App() {
          form = new JFrame();                    // Створити головне вікно
          form.setBounds(100, 100, 450, 300);   // Виставити позицію і розмір
          form.setTitle("MyWnd");
          form.setResizable(false);
          frmLogin.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

          // Отримати панель вмісту і виміряти її параметри
          Container pane = form.getContentPane();
          pane.setBackground(Color.WHITE);
          pane.setFont(new Font("Tahoma", Font.BOLD, 14));

          // Виставити розташування елементів панелі
          pane.setLayout(new FlowLayout(FlowLayout.RIGHT, 10, 10));
```

```
        // Створити кнопки
        JButton btnOK = new JButton("OK");
        // Добавити обробник
        btnOK.addActionListener(new ActionListener() {
             public void actionPerformed(ActionEvent arg0) {
                  JOptionPane.showMessageDialog(null, "Message");
             }
        });
        pane.add(btnOK);

        JButton btnCancel = new JButton("Cancel");
        btnCancel.addActionListener(new ActionListener() {
             public void actionPerformed(ActionEvent arg0) {
               System.exit(0);
             }
        });
        pane.add(btnCancel);
    }
}
```

Слухач фокуса FocusListener

Слухач FocusListener активується, коли об'єкт отримує фокус введення (тобто стає активним) або втрачає його. Концепція фокуса дуже важлива для віконних додатків. У кожен момент часу в вікні тільки один елемент управління має фокус введення – він є активним і саме він отримує інформацію про натиснуті на клавіатурі клавіші, про прокрутку коліщатка миші і т.д., і може реагувати на ці події. Користувач активує елемент управління натисканням миші або перемикаючись між елементами за допомогою клавіші Tab.

Інтерфейс FocusListener має два методи:

- public void focusGained(FocusEvent event) – викликається, коли елемент отримує фокус введення;
- public void focusLost(FocusEvent event) – викликається, коли елемент втрачає фокус введення (тобто фокус переходить до іншого елемента, або до іншого вікна).

Слухач клавіатури KeyListener

Слухач KeyListener спрацьовує для елемента управління, що має фокус введення, в момент натискання будь-якої клавіші на клавіатурі. Цей інтерфейс призначений для реагування на події низького рівня, пов'язані з натисканням окремих клавіш і їх комбінацій. Він погано підходить, наприклад, для відстеження введення окремих символів в текстове поле – для цього використовується слухач DocumentListener. Для інших типових задач, пов'язаних з клавіатурою, Swing має спеціалізовані засоби, такі як Key Bindings, Mnemonics і Accelerators, що сильно звужує область використання KeyListener. В основному він застосовується у випадках, коли необхідно управляти нестандартними (користувацькими) компонентами за допомогою клавіш.

Інтерфейс KeyListener має такі методи:

- public void mouseKeyTyped (KeyEvent event) – викликається, коли з клавіатури введено символ;
- public void mouseKeyPressed (KeyEvent event) – викликається в момент натискання клавіші (в т.ч. керуючої, не обов'язково символьної);
- public void mouseKeyReleased (KeyEvent event) – викликається в момент відпускання клавіші.

Всі ці методи отримують параметр event, який містить повну інформацію про подію. Зокрема, вираз event.getKeyChar() повертає символ (char), пов'язаний з натиснутою клавішею. Якщо з натиснутою клавішею не пов'язаний ніякий символ, повертається константа CHAR_UNDEFINED. Вираз event.getKeyCode() видає код клавіші (ціле число типу int). Цей код унікальний для всіх клавіш, включаючи керуючі, а список всіх можливих кодів визначено у вигляді констант в класі KeyEvent: VK_F1, VK_SHIFT, VK_D, VK_MINUS і т. д. Додатково

методи isAltDown (), isControlDown (), isShiftDown () дозволяють дізнатися, чи були одночасно натиснуті клавіші-модифікатори Alt, Ctrl або Shift.

Слухач зміни стану ChangeListener

ChangeListener реагує на зміну стану об'єкта. Кожен елемент управління по своєму визначає поняття «зміна стану». Наприклад, для панелі з вкладками JTabbedPane це перехід на іншу вкладку, для повзунка JSlider – зміна його положення. Для кнопки JButton зміною стану вважається кожне її натискання. Отже, хоча подія зміни стану є досить загальною, необхідно уточнювати її специфіку для кожного конкретного компонента.

В інтерфейсі визначено всього один метод:

```
public void stateChanged (ChangeEvent event);
```

Слухач подій вікна WindowListener

Слухач WindowListener може бути пов'язаний тільки з вікном (тобто контейнером верхнього рівня) і спрацьовує при різних подіях, що сталися з вікном:

- public void windowOpened(WindowEvent event) – вікно відкрилося;
- public void windowClosing(WindowEvent event) – спроба закриття вікна (наприклад, користувач натиснув на хрестик). Тут слово «спроба» означає, що метод викликається до того, як вікно буде закрито і може відмінити закриття (наприклад, вивести попередження «Ви впевнені?» і скасувати закриття вікна, якщо користувач вибере «Ні»);
- public void windowClosed(WindowEvent event) – вікно вже закрилося;
- public void windowIconified(WindowEvent event) – вікно згорнуто в панель задач;
- public void windowDeiconified(WindowEvent event) – вікно розгорнуто на екран;
- public void windowActivated(WindowEvent event) – вікно стало активним (тобто отримало фокус введення);
- public void windowDeactivated(WindowEvent event) – вікно стало неактивним.

Слухач подій компонента ComponentListener

Слухач ComponentListener оповіщається, коли спостережуваний візуальний компонент змінює своє положення, розміри або видимість. Ці зміни можуть відбуватися з різних причин, в тому числі в результаті виклику методу setLocation() або роботи менеджера розміщення. В інтерфейсі ComponentListener чотири методи:

- public void componentMoved(ComponentEvent event) – викликається, коли спостережуваний компонент переміщається;
- public void componentResized(ComponentEvent event) – викликається, коли змінюються розміри спостережуваного компонента;
- public void componentHidden(ComponentEvent event) – викликається, коли компонент стає невидимим;
- public void componentShown(ComponentEvent event) – викликається, коли компонент стає видимим.

Слухач вибору елемента ItemListener

Слухач ItemListener реагує на зміну стану спостережуваного компонента. Він застосовується не для до всіх компонентів і кожен клас компонентів по-своєму визначає поняття «зміна стану». Наприклад, для списку JComboBox цей слухач реагує на зміну обраного елемента. Для JCheckBox і JRadioButton він оповіщається при установці або знятті прапорця, для JToggleButton – при зміні стану кнопки. Слухач має один метод:

```
public void itemStateChanged(ItemEvent event).
```

Приклад – простий текстовий редактор

Створимо простий додаток для редагування довільних текстових файлів. Цей додаток дозволить:

- вводити ім'я файлу безпосередньо в текстове поле (JTextField);
- вибирати файл за допомогою стандартного діалогу відкриття файлу (JFileChooser) за кнопкою "Open";
- редагувати вміст файлу в багаторядковому текстовому полі (JTextArea);
- зберігати зміни у файлі за кнопкою "OK";
- вийти з програми за кнопкою "Cancel", відображаючи попередження перед виходом.

Вікно додатка має вигляд, показаний на рис. 6.19:

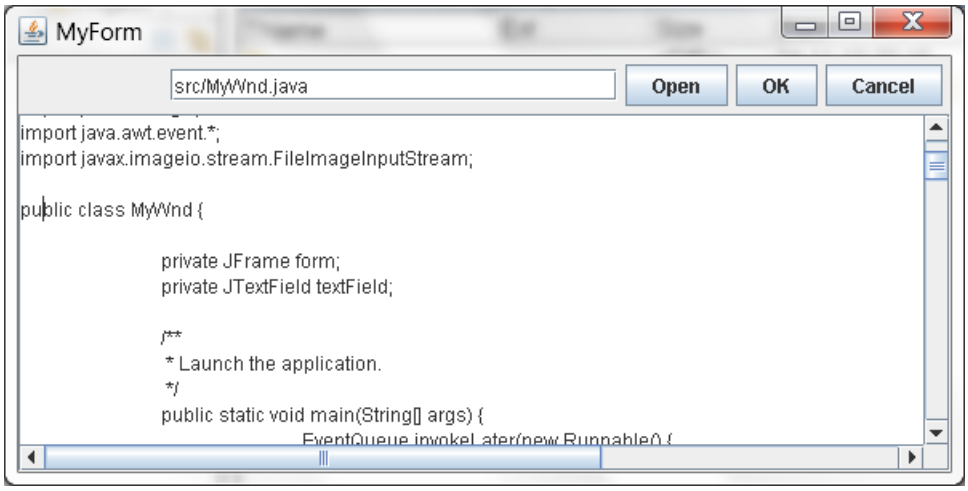

Рис. 6.19. Вікно додатку редактора

Код додатку:

```
import java.io.*;
import java.util.Scanner;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class MyApp {                    // Клас додатку

     private JFrame form;               // Головне вікно
     private JTextField txtName;        // Поле для імені файлу
     private JTextArea txtFile;         // Вміст файлу

     public static void main(String[] args) {
          MyApp app = new MyApp();      // Створити об'єкт-додаток
          app.form.setVisible(true);    // Показати його вікно
     }

     public MyApp() {
          // Створюємо вікно
          form = new JFrame();
          form.setTitle("MyForm");
          form.setBounds(100, 100, 600, 300);
          form.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

          // Розташування для елементів головної панелі
          Container pane = form.getContentPane();
          pane.setLayout(new BoxLayout(pane, BoxLayout.Y_AXIS));

          // Створюємо верхню панель
          JPanel panel = new JPanel();
```

```
panel.setLayout(new FlowLayout( FlowLayout.RIGHT ));
panel.setMaximumSize(new Dimension(600,30));
pane.add(panel);

// Добавляємо елементи верхньої панелі
txtName = new JTextField();
txtName.setColumns(25);
panel.add(txtName);
// Перевірка правильності введення
txtName.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent arg0) {
          if( !txtName.getText().matches(".+\\.html?") )
               txtName.setForeground( new Color( 255, 100, 100 ) );
          else txtName.setForeground( Color.BLACK );
     }
});
JButton btnOpen = new JButton("Open");
panel.add(btnOpen);
JButton btnOK = new JButton("OK");
panel.add(btnOK);
JButton btnCancel = new JButton("Cancel");
panel.add(btnCancel);

// Створюємо поле з прокруткою для виведення тексту
txtFile = new JTextArea("No file selected");
JScrollPane scroll = new JScrollPane (txtFile);
form.getContentPane().add(scroll);

// Додаємо обробники:
// Кнопка Open
btnOpen.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent arg0) {
          // Створюємо діалог вибору файла
          JFileChooser fc = new JFileChooser();
          // Якщо файл вибраний
          if( JFileChooser.APPROVE_OPTION ==
                    fc.showOpenDialog(form) )
               // То його ім'я записуємо  в txtName
               txtName.setText( fc.getSelectedFile().getAbsolutePath() );
     }
});
// Кнопка OK
btnOK.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent arg0) {
          try {
               // Створюємо об'єкт  File
               File f = new File(txtName.getText());
               // Перевіряємо чи файл існує
               if( !f.exists() ) txtFile.setText("No such file");
               else {
                    // Відкриваємо файл для читання з допомогою Scanner
                    Scanner sc  = new Scanner(f); String s ="";
                    // Тоді записуємо весь його зміст в поле txtFile
                    while( sc.hasNextLine() ) s = s + sc.nextLine() + "\n";
                    txtFile.setText(s);
               }
          } catch (Exception e) {
               // Виводимо повідомлення про помилку
               JOptionPane.showMessageDialog(null, e.getMessage());
          }
     }
});
// Кнопка Cancel
btnCancel.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent arg0) {
```

```
                    // Створюємо діалог підтвердження з кнопками OK і CANCEL
                    if( JOptionPane.OK_OPTION ==
                         JOptionPane.showConfirmDialog( form,
                                   "Exit without saving?", "Exit",
                                   JOptionPane.OK_CANCEL_OPTION ) )
                         // Якщо натиснуто OK, то завершуємо програму
                         System.exit(0);
               }
          });
     }
}
```

Робота з меню

Меню є невід'ємною частиною сучасних віконних додатків і представляє собою зручно згрупований набір команд. Меню поділяється на два типи: головне і контекстне. *Головне меню* розташовується уздовж верхньої межі вікна і містить команди, що відносяться до додатку в цілому (точніше, всі команди, доступні в даному вікні додатку). *Контекстне меню* викликається натисненням правої кнопки миші на конкретному графічному елементі і містить команди, які можуть бути застосовані саме до цього елемента.

Головне меню

В Swing *головне меню* вікна представлено класом ***JMenuBar***. Меню являє собою панель з менеджером розташування BoxLayout (по горизонталі), в яку можна додавати як пункти меню, так і будь-які інші графічні компоненти: випадні списки, кнопки, навіть панелі з закладками. Однак для зручності користувача рекомендується використовувати тільки «традиційні» пункти меню.

Головне меню приєднується до вікна методом `setJMenuBar(JMenuBar menuBar)`.

Елементами головного меню є *звичайні меню*, які являють собою випадні прямокутні блоки команд і є об'єктами класу ***JMenu***. Конструктор JMenu(String title) приймає один параметр: назву меню, яка буде відображатися в рядку головного меню.

Кожне меню у свою чергу складається з *пунктів меню*, представлених класом ***JMenuItem***. За логікою роботи пункти меню аналогічні кнопці JButton, тобто при натисканні на кожному пункті виконується певна дія.

Елемент меню створюється порожнім конструктором JMenuItem() або одним з конструкторів, в які передається текст і/або значок елемента меню:

- JMenuItem(String text),
- JMenuItem(Icon icon),
- JMenuItem(String text, Icon icon).

В будь-який момент текст і значок можна змінити методами setText(String text) і setIcon(Icon icon) відповідно.

Запис додається до меню JMenu методом add(JMenuItem item). Щоб відокремити групи взаємопов'язаних елементів меню можна додати між ними роздільник методом addSeparator().

Також в меню можна додати вкладене меню. Вкладені меню дуже широко використовуються в сучасних додатках, але зазвичай глибина вкладеності більше трьох рівнів незручна для користувачів.

Для прикладу додамо у вікно програми головне меню, що складається з двох підменю: «Файл» і «Правка», а в меню «Правка» помістимо випадне підменю (рис. 6.20). Скористаємося знаннями про менеджер розташування головного меню щоб додати з правого краю значок (який не є пунктом меню).

```
SimpleWindow() {
    super("Вікно з меню");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    JMenuBar menuBar = new JMenuBar();
    JMenu fileMenu = new JMenu("Файл");
    fileMenu.add(new JMenuItem("Новий"));
    fileMenu.add(new JMenuItem("Відкрити", new ImageIcon("1.png")));
```

```
    fileMenu.add(new JMenuItem("Зберегти"));
    fileMenu.addSeparator();
    fileMenu.add(new JMenuItem("Вийти"));
    JMenu editMenu = new JMenu("Правка");
    editMenu.add(new JMenuItem("Копіювати"));
    JMenu pasteMenu = new JMenu("Вставити");
    pasteMenu.add(new JMenuItem("Із буфера"));
    pasteMenu.add(new JMenuItem("Із файлу"));
    editMenu.add(pasteMenu);
    menuBar.add(fileMenu);
    menuBar.add(editMenu);
    menuBar.add(Box.createHorizontalGlue());
    menuBar.add(new JLabel(new ImageIcon("2.png")));
    setJMenuBar(menuBar);
    setSize(250, 150);
}
```

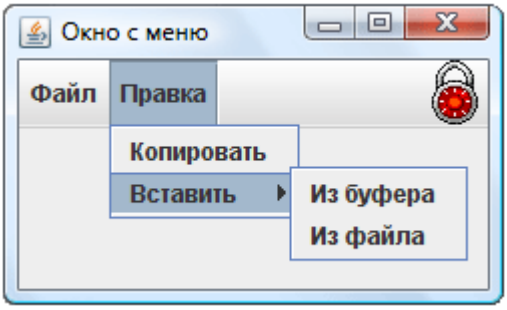

Рис. 6.20. Вікно з головним меню

Обробка команд меню

Кожен елемент меню по суті являє собою кнопку, тому реагує на такі ж події і дозволяє приєднувати до себе таких же слухачів, що і кнопка. Наприклад, щоб при виборі в меню елемента «Вийти» програма припиняла свою роботу, слід замінити в прикладі рядок

```
fileMenu.add(new JMenuItem("Вийти"));
```

на наступні рядки:

```
JMenuItem exit = new JMenuItem("Вийти");
exit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent event) { System.exit(0); }
});
fileMenu.add(exit);
```

Зауважимо, що метод `System.exit(0)` закінчує роботу додатку.

Як створити контекстне меню

Контекстне (інакше спливаюче) меню реалізовано в класі **JPopupMenu**, дуже схожому на клас JMenu. Його відмінністю є метод

```
show(Component comp, int x, int y),
```

який відображає меню в точці з заданими координатами щодо меж батьківського компонента.

Контекстне меню, як правило, відображається при натисканні правою кнопкою миші над батьківським компонентом. Отже, щоб відобразити меню, потрібно додати до цього компоненту слухача миші.

Розглянемо приклад, в якому до вже створеного вікна додається напис із контекстним меню (рис. 6.21). В конструкторі вікна необхідно додати наступний код:

```
label=new JLabel("КНИЖКА", new ImageIcon("book.png"), JLabel.RIGHT);
JPanel panel = new JPanel();
panel.add(label);
popup=new JPopupMenu();

popup.add(new JMenuItem("Прочитати"));
popup.add(new JMenuItem("Спалити") );
label.addMouseListener(new MouseAdapter() {
    public void mouseClicked (MouseEvent event){
        if (SwingUtilities.isRightMouseButton(event)) {
            popup.show(label, event.getX(), event.getY());
        }
    }
});
setContentPane(panel);
```

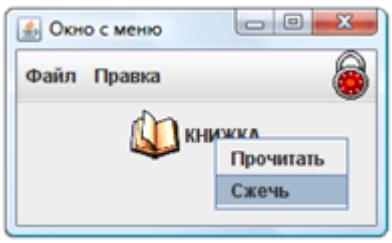

Рис. 6.21. Контекстне меню для напису

При цьому в класі вікна потрібно додати два поля:

```
JPopupMenu popup; JLabel label;
```

Це необхідно для того, щоб анонімний клас-слухач міг звернутися і до мітки і до контекстного меню.

Наведений приклад також ілюструє застосування методу `isRightMouseButton (MouseEvent event)` у слухачі подій миші. Цей метод визначений в класі допоміжних утиліт SwingUtilities і дозволяє за подією MouseEvent визначити, чи була натиснута саме права кнопка миші. При цьому методи event.getX() і event.getY() повертають координати курсора миші відносно спостережуваного компонента.

Клас AbstractAction та інтерфейс Action

Дуже часто одна і та ж команда дублюється в графічному інтерфейсі і може бути викликана кількома шляхами. Скажімо, команда «Копіювати» може бути присутня в головному меню вікна, в контекстному меню певного об'єкту, а також у вигляді кнопки на панелі інструментів. Це необхідно для зручності користування програмою. Проте, прив’язувати до всіх згаданих компонентів (кнопок і команд меню) однакових слухачів досить незручно. Більш того, всі ці команди однаково повинні реагувати на зміни стану програми: якщо копіювання з якихось причин стає неможливим, то всі елементи управління, що представляють дану команду, повинні стати неактивними.

Для спрощення роботи в таких ситуаціях передбачений інтерфейс Action. Цей інтерфейс збирає в одному місці все, що відноситься до деякої дії: обробник, показник активності, підказку, значок і т. д.

Інтерфейс Action успадкований від інтерфейсу ActionListener, тому його головним методом є `actionPerformed(ActionEvent event)`. Саме тут програмується необхідна дія. Крім того, за допомогою методу `putValue(String key, Object value)` можна задати додаткові властивості дій. Перший параметр – це строковий ідентифікатор властивості. Цей параметр може приймати одне із значень, описаних в константах інтерфейсу Action. Другий параметр – об'єкт, який представляє собою значення властивості.

Основні властивості інтерфейсу Action та відповідні їм константи:

- `NAME` – ім'я дії,
- `SMALL_ICON` – значок, що відповідає даній дії,
- `SHORT_DESCRIPTION` – короткий опис дії.

Зробити дію активною чи неактивною дозволяє метод `setEnabled(boolean enabled)`.

На основі об'єкта-дії можна створювати різні елементи управління, передаючи цю дію в якості єдиного параметра конструктора. Елементи меню і кнопки, зокрема, якраз і відносяться до таких елементів управління.

Недоліком інтерфейсу Action є занадто велика кількість допоміжних абстрактних методів, які досить складно програмувати (наприклад, методи setEnabled(), putValue()). Тому зазвичай замість нього використовується клас AbstractAction, який містить реалізації за замовчанням всіх методів інтерфейсу Action, крім actionPerformed(). Тоді нам залишається написати код тільки для одного метода.

Розглянемо приклад.

```
public class SimpleWindow extends JFrame {
    private ExitAction exitAction;

    SimpleWindow() {
        super("Вікно з меню");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        exitAction = new ExitAction();
        DeactivateAction deactivateAction = new DeactivateAction();
        JMenuBar menuBar = new JMenuBar();
        JMenu fileMenu = new JMenu("Файл");
        fileMenu.add(new JMenuItem("Новий"));
        fileMenu.addSeparator();
        fileMenu.add(deactivateAction);
        fileMenu.add(exitAction);
        menuBar.add(fileMenu);
        setJMenuBar(menuBar);
        JToolBar toolBar = new JToolBar("Панель інструментів");
        toolBar.add(exitAction);
        toolBar.add(deactivateAction);
        getContentPane().add(toolBar, BorderLayout.NORTH);
        JPanel panel = new JPanel();
        panel.add(new JButton(exitAction));
        panel.add(new JButton(deactivateAction));
        getContentPane().add(panel);
        setSize(250, 250);
    }

    class ExitAction extends AbstractAction {
        ExitAction() {
            putValue(Action.NAME, "Вийти");
            putValue(Action.SHORT_DESCRIPTION, "Завершення програми.");
            putValue(Action.SMALL_ICON, new ImageIcon("2.png"));
        }

        public void actionPerformed(ActionEvent event) {
            System.exit(0);
        }
    }

    class DeactivateAction extends AbstractAction {
        DeactivateAction() {
            putValue(Action.NAME, "Заборонити вихід");
            putValue(Action.SMALL_ICON, new ImageIcon("1.png"));
        }

        public void actionPerformed(ActionEvent event) {
            if (exitAction.isEnabled()) {
```

```
                exitAction.setEnabled(false);
                putValue(Action.NAME, "Дозволити вихід");
            } else {
                exitAction.setEnabled(true);
                putValue(Action.NAME, "Заборонити вихід");
            }
        }
    }
}
```

Тут в класі вікна описані два вкладених класи-дії. Перша дія – це вихід з програми. Об'єкт цього класу exitAction є полем в класі вікна і на його основі створюються пункт меню та дві кнопки. В результаті пункт меню і обидві кнопки будуть запускати одну і ту саму дію, мати однакові назви, значки та спливаючі підказки. Другий клас-дія активує і деактивує об'єкт exitAction, роблячи першу дію доступною або навпаки недоступною. При цьому також змінюється назва всіх кнопок, що запускають другу дію. Запустіть приклад і поспостерігайте за тим, як виглядають і поводяться всі кнопки і пункти меню в обох випадках: до вибору дії «Заборонити вихід» і після неї.

Стандартні діалогові вікна

Клас JOptionPane

Невід'ємною частиною більшості програм є невеликі діалогові вікна, які з'являються на екрані за потребою. Зазвичай вони виводять користувачеві повідомлення (наприклад, повідомлення про помилку), задають питання (наприклад, вікно підтвердження або скасування дії), або вимагають певну інформацію (наприклад, вікно відкриття файлу). Подібні діалогові вікна можна запрограмувати вручну на основі класу JDialog, який багато в чому схожий на знайомий нам JFrame. Однак з огляду на те, що ці вікна є типовими для багатьох додатків, Swing надає в розпорядження програміста кілька готових класів для роботи з ними.

Найчастіше використовується клас **JOptionPane**, що містить кілька статичних методів для створення стандартних діалогових вікон.

Метод `showMessageDialog()` виводить вікно-повідомлення, яке містить напис, значок і кнопку OK. Існує кілька різновидів цього методу з різними наборами параметрів. Найпростіший з них showMessageDialog(Component component, Object content) вимагає вказати компонент, над яким з'явиться діалогове вікно і вміст вікна. Найчастіше вмістом вікна є рядок повідомлення. Замість першого параметра можна передати null – тоді вікно з'являється по центру екрана. Розширений варіант showMessageDialog(Component component, Object content, String title, int type) дозволяє задати також заголовок вікна і тип повідомлення (від якого залежить іконка у вікні): повідомлення про помилку (ERROR_MESSAGE), попередження (WARNING_MESSAGE), інформація (INFORMATION_MESSAGE).

Діалогове вікно є ***модальним***. Це означає, що поки користувач не натисне в цьому вікні кнопку OK, додаток буде заблоковано – користувач не зможе працювати з іншими вікнами.

Наведемо приклад створення вікна-попередження (рис. 6.22).

```
public class SimpleWindow extends JFrame {
    private JButton button;

    SimpleWindow() {
       super("Попереджувальний діалог");
       setDefaultCloseOperation(EXIT_ON_CLOSE);
       button = new JButton("Інформація");
       button.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent event) {
               JOptionPane.showMessageDialog(button,
                   "Не треба було натискати на цю кнопку", "Інформація",
                   JOptionPane.WARNING_MESSAGE);
           }
       });
```

```
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(button);
        setSize(200, 150);
    }
}
```

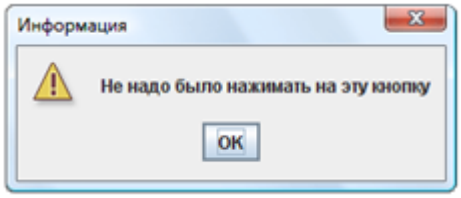

Рис. 6.22. Вікно попередження

Часто від користувача вимагається певна відповідь, в найпростішому випадку – відповідь «так» чи «ні» – тоді використовується діалогове вікно “Confirm Dialog” (вікно підтвердження). Воно має кілька кнопок, одну з яких користувач повинен натиснути. Програма тоді повинна отримати інформацію про вибір користувача, на основі якої і будується подальша логіка її роботи.

Вікно підтвердження відображається методом `showConfirmDialog(Component component, Object content)`. Параметри цього методу є ідентичними за змістом параметрам showMessageDialog(), але тепер вікно має не одну кнопку, а три: «Yes», «No» і «Cancel». Метод повертає значення, вибране користувачем, яке можна порівняти з константами YES_OPTION, NO_OPTION і CANCEL_OPTION.

Покажемо логіку роботи з цим методом на прикладі (рис. 6.23)

```
public class SimpleWindow extends JFrame {
    JButton button;

    SimpleWindow() {
        super("Діалог підтвердження");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        button = new JButton("Вихід");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                if (JOptionPane.showConfirmDialog(button,
                        "Ви впевнені,що хочете вийти?") == JOptionPane.YES_OPTION) {
                    System.exit(0);
                }
            }
        });
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(button);
        setSize(200, 150);
    }
}
```

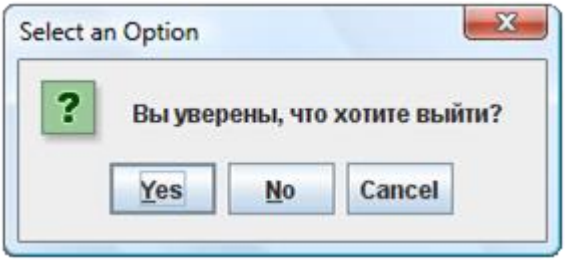

Рис. 6.23. Вікно підтвердження

Метод showConfirmDialog() має ще ще чотири різновиди з різним набором параметрів, які дозволяють змінювати заголовок і значок вікна, а також набір кнопок.

Клас JFileChooser

Swing має готове вікно для вибору файлу, яке корисне, наприклад, для програмування пункту меню «Файл» ® «Відкрити». Об'єкт класу **JFileChooser** створюється простим конструктором без параметрів, після чого може виводитися на екран методом `showOpenDialog()`. Цей метод повертає результат, яким завершився вибір файлу і який має наступні можливі значення (описані як константи в класі JFileChooser):

`APPROVE_OPTION` – вибір файлу пройшов успішно; тепер можна методом getSelectedFile() отримати обраний файл;

`CANCEL_OPTION` – користувач скасував вибір файлу, натиснувши кнопку Cancel;

`ERROR_OPTION` – при виборі файлу сталася помилка або користувач закрив діалогове вікно хрестиком.

Метод `showSaveDialog()` відображає схоже вікно, призначене для збереження файла. Користувач може не обирати існуючий файл, а вибрати директорію для збереження файлу і вказати ім'я нового файлу. Метод повертає результат того ж типу, що й showOpenDialog(). Якщо вибір шляху для збереження пройшов успішно, виклик методу getSelectedFile() поверне об’єкт File, що містить повне ім’я файлу (зі шляхом), куди користувач бажає зберегти дані.

Слід мати на увазі, що сам клас JFileChooser не відкриває і не зберігає файлів. Він тільки повертає шлях до обраного користувачем файлу. А відкрити файл або зберегти його в заданому місці повинна вже сама програма.