Лабораторна робота № 8

Розробка додатків з графічним інтерфейсом користувача

Мета роботи: Вивчити способи розробки додатків Java з графічним інтерфейсом користувача з використанням бібліотек AWT і Swing.

1. Теоретичні відомості

Додаток з *графічним інтерфейсом користувача* (*Graphic User Interface - GUI*) - це додаток, в якому застосовуються класи і інтерфейси, що представляють собою набори *компонентів* або *віджетів* - багаторазово використовуваних класів і інтерфейсів, що формують стандартні елементи управління графічного інтерфейсу користувача та виконують стандартну обробку взаємодії користувача з ними. Платформа Java містить класичні засоби побудови графічного інтерфейсу користувача - бібліотеки *AWT* (*Abstract Window Toolkit*) і *Swing*, а також засоби побудови додатків з сучасним інтерфейсом - так званих *додатків з насиченим інтерфейсом* (*Rich Internet Application - RIA*) - *JavaFX*, які можуть запускатися як настільні додатки, додатки браузерів або мобільні додатки. Кожна із зазначених бібліотек надає багатий набір візуальних компонентів, наприклад, форм, написів, кнопок, текстових полів, прапорців та інших компонентів, розроблених так, щоб їх можна було успішно застосовувати в самих різних додатках.

Технологія Swing з'явилася як спроба подолання проблем історично першої бібліотеки - AWT. В AWT був визначений базовий набір елементів управління і вікон, що дозволяє створювати графічні інтерфейси. Суттєвим обмеженням AWT була платформово-орієнтована підтримка візуальних компонентів, тобто засоби AWT просто викликали функції API операційної системи для створення візуального компонента. В результаті зовнішній вигляд інтерфейсних елементів визначався не засобами Java, а використовуваною платформою, і відрізнявся для різних операційних систем. Елементи, що створюються засобами AWT, називалися *великоваговими*. Всі компоненти Swing, за невеликим винятком, є *легковагими*, їх зовнішній вигляд визначає Swing, а не операційна система. Тому інтерфейсні елементи, створені за допомогою Swing, виглядають однаково на всіх платформах. Вони можуть мати форму, відмінну від прямокутної, змінювати прозорість і т.д. Таким чином, такі компоненти є більш ефективними і гнучкими. Оскільки кожен компонент відтворюється за допомогою Java-коду, його зовнішній вигляд повністю контролюється засобами Swing. Це означає, що зовнішній вигляд компонента можна переналаштувати в залежності від результатів виконання операторів програми. Більш того, можна створювати *глобальні стилі* (*look and feel - L&F*), що визначають, як буде виглядати інтерфейс в цілому. При перемиканні стилю зовнішній вигляд всіх елементів змінюється автоматично. його зовнішній вигляд повністю контролюється засобами Swing.

Незважаючи на те, що Swing усуває ряд обмежень, властивих AWT, він не замінює даний інструмент. Оскільки обробка подій взаємодії користувача з візуальними компонентами інтерфейсу ніяк не залежить від зовнішнього вигля-

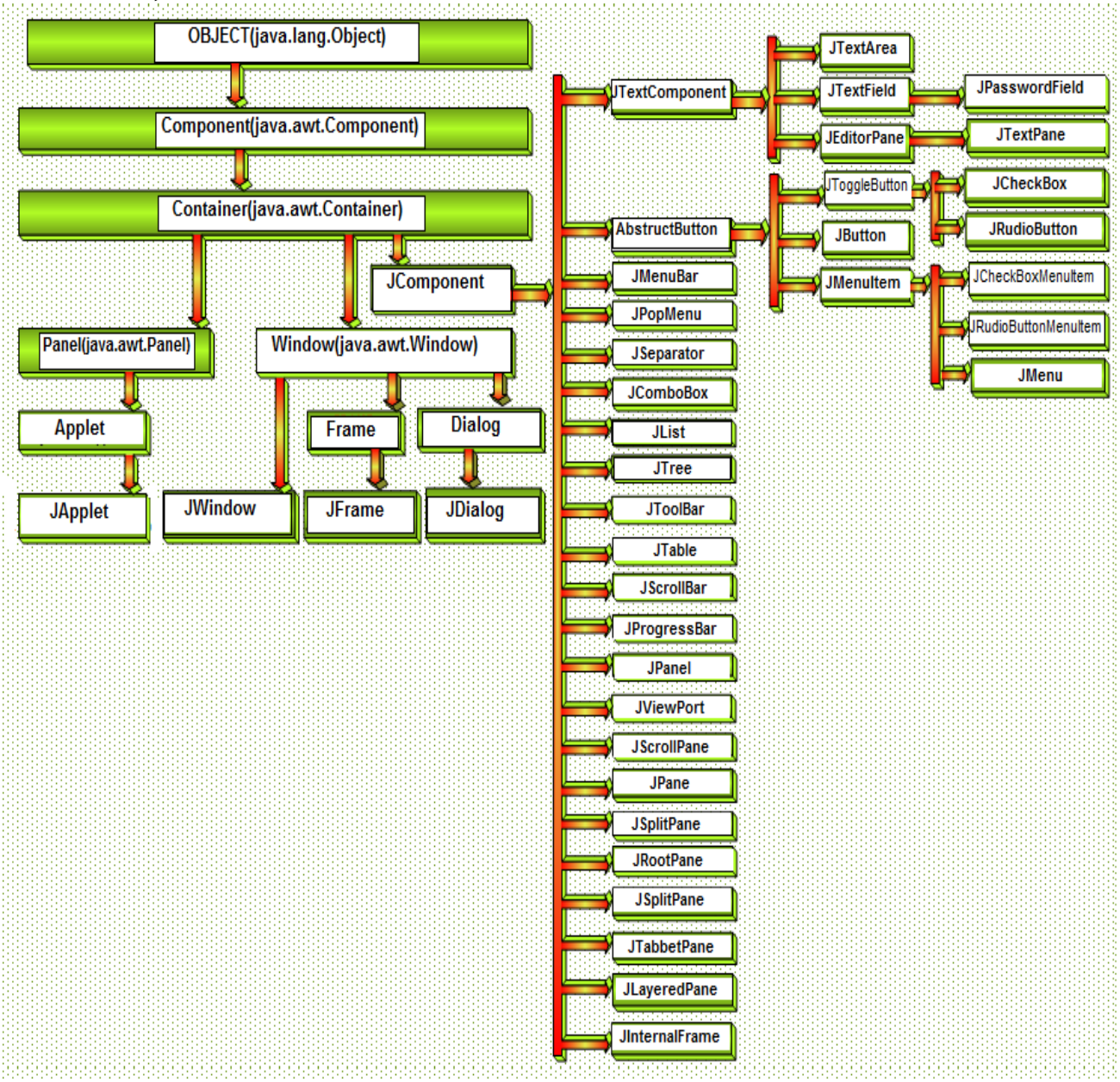
ду останніх, в GUI-додатках для побудови графічного інтерфейсу використовують компоненти Swing, а для обробки подій взаємодії з користувачем - засоби AWT.

До складу графічного інтерфейсу користувача, створеного засобами Swing, входять елементи двох типів: *компоненти* і *контейнери*. Такий поділ багато в чому умовний, так як контейнери є в той же час і компонентами. Різниця між ними в їх призначенні. *Компоненти* - це незалежні елементи, наприклад, кнопки або текстові поля. *Контейнер* може містити в собі кілька компонентів (та контейнерів) і являє собою спеціальний тип компонента. Прикладами контейнерів є форми, панелі, вкладки та ін. Щоб компонент відобразився на екрані, його необхідно помістити в контейнер. Таким чином, у складі GUI-додатка повинен бути присутнім хоча б один контейнер. Оскільки контейнери є компонентами, один контейнер може перебувати в складі іншого. Це дозволяє формувати так звану ієрархію контейнерів, на вершині якої повинен знаходитися *контейнер верхнього рівня*. Контейнери Swing організовуються з використанням шаблону проектування *Компоновник* (*Composite*), який дозволяє групі об'єктів бути представленою у вигляді єдиного об'єкта з подібними властивостями, підтримуючи при цьому однаковість взаємодії з групою і об'єктами, що входять до неї.

Класи, що представляють всі компоненти Swing, містяться в пакеті javax.swing. Переважна більшість компонентів Swing створюється за допомогою класів, які є нащадками абстрактного класу JComponent (Мал. 1) (Для компонентів Swing було прийнято рішення давати їм імена, що починаються з великої літери J). Клас JComponent реалізує функціональні можливості, загальні для всіх компонентів, наприклад, містить список обробників подій даного компонента і підтримує стилі. JComponent успадковує властивості класів AWT java.awt.Container і java.awt.Component. Таким чином, компоненти Swing будуються на базі AWT-компонентів і сумісні з ними.

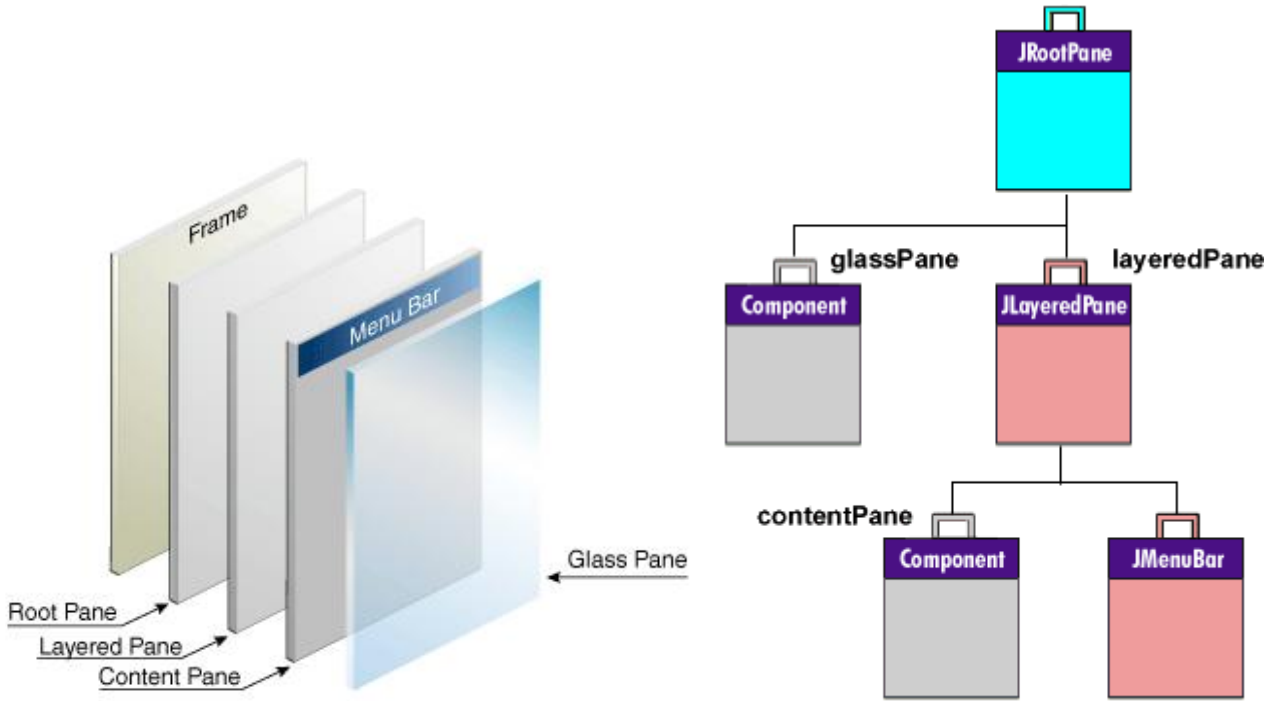
У Swing визначені два типи контейнерів. До першого типу відносяться *контейнери верхнього рівня*: JApplet, JWindow, JFrame і JDialog. Вони не належать до числа підкласів JComponent, тим не менш, є нащадками java.awt.Container і java.awt.Component. На відміну від інших компонентів Swing, контейнери верхнього рівня не можуть включатися до складу інших контейнерів і повинні перебувати на вершині ієрархії компонентів GUI-додатку, тобто бути контейнерами для всіх інших компонентів програми. Для настільних додатків в якості такого контейнера використовують об'єкт класу JFrame, для виконуваних браузерами додатків, інтегрованих у Веб-сторінки - об'єкт класу JApplet. Об'єкти класу JWindow є контейнери-вікна, які не містять заголовка з кнопками управління вікном, а об'єкти класу JDialog дозволяють створювати модальні і немодальні діалогові вікна. Контейнери другого типу - це *легковагові контейнери*, які є нащадками JComponent. Як приклади легковажних контейнерів можна привести JPanel (Панель) і JScrollPane (Прокрутка панелі), які використовуються для групування компонентів, JInternalFrame, що дозволяє створювати вікна-

контейнери з заголовками з кнопками управління вікном всередині великоваго-вого JFrame, та ін.



Мал. 1. Ієрархія класів Swing

У кожному контейнері верхнього рівня, а також легковаговому контейнері JInternalFrame реалізований набір панелей, представлений на Мал. 2. На вершині ієрархії знаходиться коренева панель - екземпляр класу JRootPane, автоматично створювана при створенні зазначених вище контейнерів. Це легковаговий контейнер, мета якого - управління іншими панелями і, при необхідності, меню (що зазвичай містить команди для роботи з додатком).



Мал. 2. Структура контейнера Swing верхнього рівня

Коренева панель включає в себе *скляну панель* (*Glass Pa*ne), *панель шару* (*Layered Pane*) і *панель вмісту* (*Content Pane*). *Скляна панель* - це прозора панель верхнього рівня, яка розташована поверх інших панелей і заповнює всю область кореневої панелі. *Скляна панель* (*Glass Pane*) дозволяє управляти подіями миші, що відносяться до всього контейнера, а не до компонентів, які містяться в ньому.

Панель шару (*Layered Pane*) являє собою екземпляр класу JLayeredPane, також заповнює всю область кореневої панелі. Вона підтримує "третій вимір" для компонентів, тобто визначає правила перекриття одних компонентів іншими. У складі панелі шару знаходиться *панель вмісту* і може також перебувати *рядок меню*. Незважаючи на те, що скляна панель і панель шару - невід'ємні частини контейнера верхнього рівня і виконують важливі функції, їх дії здебільшого приховані не тільки від користувачів, але і від розробників.

Додаток в основному взаємодіє з *панеллю вмісту* (*Content Pane*), оскільки саме в неї включаються візуальні компоненти. Іншими словами, додаючи компонент, наприклад кнопку, до контейнера верхнього рівня, Ви насправді додаєте його до панелі вмісту. За замовчуванням панель вмісту являє собою "непрозорий екземпляр" JPanel.

Розглянемо як приклад просту програму, створену з використанням засобів Swing (Мал. 3). Дана програма демонструє ключові властивості Swing і дає уявлення про контейнер верхнього рівня JFrame (Форма або фрейм), що зазвичай використовуються в настільних додатках, і компоненті JLabel (Напис), за допомогою якого можливе виведення текстової інформації на формі. У даній програмі створюється контейнер JFrame, в який поміщається екземпляр компонента JLabel, що відображає текстове повідомлення:

```
package swingapppackage;

/*Подключение библиотеки классов Swing*/
import javax.swing.*;

public class SwingDemo {

    /*Создание формы в конструкторе*/
    public SwingDemo() {

        /*Создание контейнера верхнего уровня с заданием текста
          заголовка окна*/
        JFrame jfrm = new JFrame("Простая программа Swing");

        /*Установка начальных размеров формы*/
        jfrm.setSize(370, 100);

        /*Конфигурирование завершения работы программы
          при закрытии пользователем окна приложения*/
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /*Создание надписи*/
        JLabel jlab = new JLabel("Swing - основа современных Java "
                                 + "GUI!");

        /*Включение надписи в форму-контейнер*/
        jfrm.add(jlab);

        /*Отображение формы*/
        jfrm.setVisible(true);
    }

    /*Главный метод приложения*/
    public static void main(String args[]) {
        /*Создание формы в потоке, отдельном от потока
          обработки событий*/
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                SwingDemo swingDemo = new SwingDemo();
            }
        });
    }
}
```

Мал. 3. Простий додаток Swing

На початку програми здійснюється імпортування пакета javax.swing, який містить засоби Swing. Зокрема, в ньому визначені класи, що реалізують візуальні компоненти і контейнери. Цей пакет повинен бути включений в кожну програму, яка використовує Swing.

```
import javax.swing. *;
```

Далі в програмі оголошується клас `SwingDemo` і його конструктор. Саме в конструкторі виконується велика частина дій програми. Код конструктора починається зі створення об'єкта-контейнера `JFrame` з ім'ям `jfrm`:

```
JFrame jfrm = new JFrame("Проста програма Swing");
```

Контейнер `jfrm` визначає прямокутне вікно, що містить заголовок, в який виводиться рядок, зазначений як параметр конструктора, кнопки, призначені для закриття, мінімізації, максимізації і відновлення розмірів вікна. Потім за допомогою методу `setSize`, який клас JFrame успадковує від класу java.awt.Component, встановлюються горизонтальний і вертикальний розміри вікна форми в пікселах.

```
jfrm.setSize(370, 100);
```

За замовчуванням при закритті вікна контейнера верхнього рівня (для цього призначена кнопка у верхньому правому куті) вікно видаляється з екрану, але програма не завершує свою роботу. Зазвичай бажано, щоб при закритті вікна контейнера верхнього рівня додаток завершував роботу. Зробити це можна викликом методу класу JFrame `setDefaultCloseOperation()` із зазначенням для нього параметра `JFrame.EXIT_ON_CLOSE`:

```
jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Замість значення параметра `JFrame.EXIT_ON_CLOSE` методу можна передавати і інші значення - `JFrame.DISPOSE_ON_CLOSE`, `JFrame.HIDE_ON_CLOSE` `JFrame.DO_NOTHING_ON_CLOSE`. Імена констант відображають їх призначення. Всі вони визначені в інтерфейсі javax.swing.WindowConstants, що реалізується класом JFrame.

Наступний рядок коду створює Swing-компонент JLabel (Напис) з ім'ям `jlab`:

```
JLabel jlab = new JLabel("Swing - основа сучасних Java GUI!");
```

Рядок, зазначений як параметр конструктора класу JLabel, буде відображений в написі на формі.

Наступний рядок коду додає напис до контейнера `jfrm`:

```
jfrm.add(jlab);
```

Слід зазначити, що метод додавання компонентів безпосередньо контейнеру з'явився, починаючи з Java 5, до цього необхідно було явно вказувати додавання компонента до панелі вмісту контейнера, в цьому випадку розглянутий вище оператор виглядав би так:

```
jfrm.getContentPane().add(jlab);
```

Метод `getContentPane()` повертає посилання на панель вмісту форми. Незважаючи на відсутність явної вказівки в першому операторі (що використовується, починаючи з Java 5), компоненти за замовчуванням додаються саме до панелі вмісту контейнера.

Останній оператор конструктора `SwingDemo()` забезпечує відображення вікна.

```
jfrm.setVisible(true);
```

Метод `setVisible()` успадкований від java.awt.Component, він має такий вигляд:

```
void setVisible(boolean flag)
```

Якщо значення параметра `flag` є `true`, вікно відображається на екрані. В іншому випадку воно залишається прихованим. За замовчуванням фрейм невидимий, тому для його відображення треба викликати метод `setVisible(true)`.

У методі `main()` створюється об'єкт класу `SwingDemo`, в результаті чого вікно з написом з’являється на екрані.

```
SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                SwingDemo swingDemo = new SwingDemo();
            }
        });
```

Наведений вище фрагмент коду створює об'єкт `swingDemo` не в основному потоці виконання (thread), а в окремому потоці, який створюється за допомогою анонімного класу, що реалізовує інтерфейс java.lang.Runnable. Такий підхід дозволяє розділити потік, в якому вимальовується графічний інтерфейс користувача від потоку, в якому виконуються методи обробки подій взаємодії користувача з візуальними компонентами. Утилітарний клас javax.swing.SwingUtilities підтримує два методи створення додаткового потоку: `invokeLater()` та `invokeAndWait()`:

```
static void invokeLater(Runnable obj)
static void invokeAndWait(Runnable obj)
       throws InterruptedException, InvocationTargetException
```

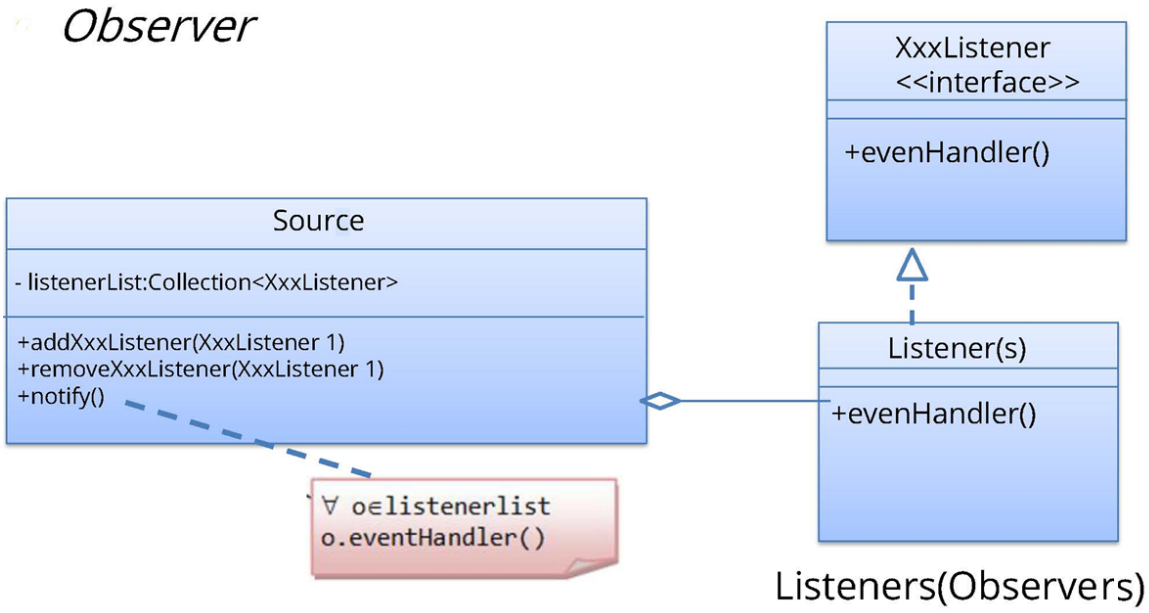
Різниця між двома методами, представленими вище, полягає в тому, що `invokeLater()` відразу ж повертає керування методу, який його викликає, а `invokeAndWait()` очікує завершення методу `obj.run()`.

Необхідно відзначити ще одну особливість розглянутої вище програми - вона не реагує на дії користувача, оскільки компонент JLabel використовують тільки для виведення текстової інформації і при цьому зазвичай відсутня необхідність взаємодії компонента з користувачем (хоча, як і для будь-якого компонента, її можна організувати). З цієї причини в програму не були включені *методи обробки подій* взаємодії з користувачем. Однак існують компоненти, які зазвичай генерують *події* взаємодії з користувачем, у відповідь на які програма повинна виконувати певні дії. Наприклад, події генеруються клацанням на кнопці миші, натисканням клавіші на клавіатурі або при виборі елемента списку. Існують події, безпосередньо не пов'язані з діями користувачів, наприклад, генерування подій після закінчення інтервалу часу таймера. Незалежно від при-

чини виникнення тієї чи іншої події, засоби, які перехоплюють і обробляють їх, є важливою частиною будь-якого Swing-додатку.

Як зазначалося вище, в Swing використовується той же механізм обробки подій, що і в AWT. Він носить назву *модель делегування подій* і реалізований з використанням шаблону проектування *Оглядач* (*Observer*) (Мал. 4). *Джерело* (*Source* - зазвичай візуальний компонент) при взаємодії з користувачем генерує об'єкт-*подію* (*Event*), який передається об'єктам-*слухачам події* (*Listener*), посилання на які додані в список слухачів об'єкта-джерела, і викликає з об'єктів-слухачів метод обробки події. В рамках даної моделі слухачі лише очікують виникнення події. При отриманні об'єкта-події вони обробляють його (виконують якісь необхідні в програмі дії) і повертають управління. Перевага такого підходу в тому, що логіка обробки подій відділена від логіки побудови інтерфейсу користувача, який генерує ці події. Елемент інтерфейсу "делегує" обробку події окремому фрагменту коду.

Згідно з моделлю делегування, подія є об'єктом, що описує зміни стану джерела. Подія може бути наслідком дій користувача з елементом графічного інтерфейсу або згенерована програмними засобами. Суперкласом всіх подій є java.util.EventObject. Багато подій оголошені в пакеті java.awt.event, але деякі містяться в javax.swing.event.



Мал. 4. Модель обробки подій в Swing

Реєстрація слухачів подій в Swing здійснюється шляхом виклику методу `addТипListener(ТипListener l)` класу-джерела події (компонента), де `Тип` - це ім'я події, а параметр `l` являє собою посилання на слухач з методом обробки події. Для кожного типу події визначений власний слухач. Наприклад, метод, який додає слухач подій клавіатури, називається `addKeyListener(KeyListener l)`. Для додавання слухача подій, пов'язаних з переміщенням миші, використовується метод `addMouseMotionListener(MouseMotionListener l)` і т.і.

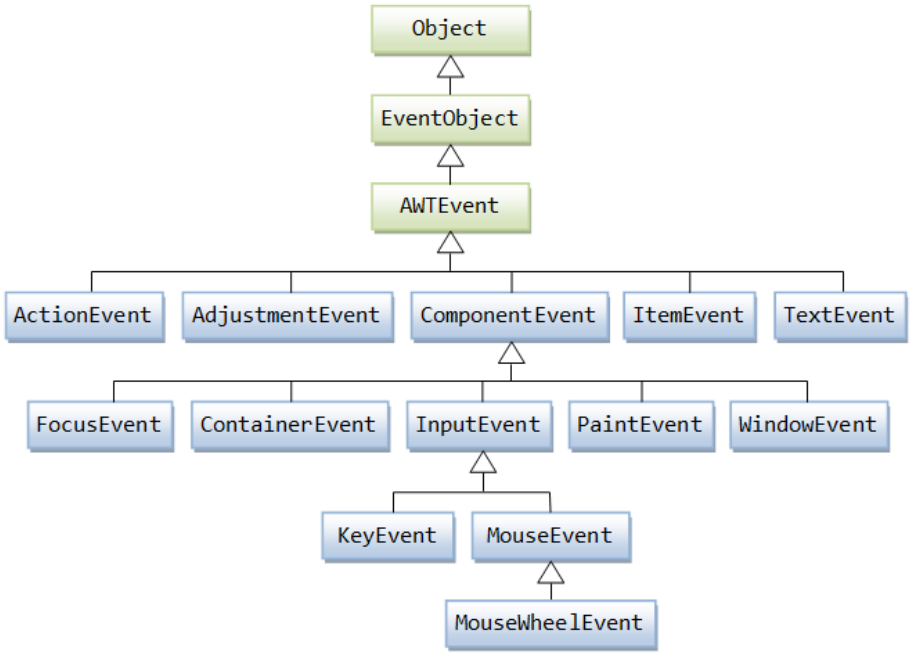
Джерело (компонент) має також метод, що дозволяє видалити слухача подій певного типу зі списку слухачів джерела:

```
public void removeТипListener(TunListener el)
```

Наприклад, для того, щоб видалити слухача подій клавіатури, треба викликати метод `removeKeyListener(KeyListener l)`.

Методи, що дозволяють отримувати і обробляти події, визначені в інтерфейсах, що містяться в пакетах java.awt.event, javax.swing.event і java.beans. Наприклад, в інтерфейсі ActionListener оголошений метод, який викликається тоді, коли користувач виконує клацання мишею на кнопці або виконує іншу дію, яка зачіпає компонент.

Класи, що представляють події, лежать в основі механізму обробки подій Java. Ці класи формують ієрархічну структуру, на вершині якої знаходиться клас java.util.EventObject (Мал. 5). Він є суперкласом для всіх подій. Абстрактний клас java.awt.AWTEvent є підкласом EventObject. Він, в свою чергу, є батьківським класом для всіх класів-подій AWT, використовуваних в моделі делегування подій. Swing використовує події AWT і, крім того, визначає кілька додаткових подій, визначених в пакеті javax.swing.event.



Мал. 5. Ієрархія подій, які використовуються в AWT і Swing

В Табл. 1 описані деякі класи подій і відповідні їм інтерфейси слухачів, визначені в пакетах javax.awt.event і javax.swing.event.

Табл. 1. Деякі класи подій з пакетів java.awt.event і javax.swing.event

Клас події	Опис	Слухач
java.awt.event		
ActionEvent	Генерується при виконанні дій з інтерфейсним елементом, наприклад після клацання на кнопці	ActionListener
AdjustmentEvent	Генерується при виконанні дій зі смугою	AdjustmentListener

	прокрутки	
FocusEvent	Генерується тоді, коли компонент отримує або втрачає фокус введення	FocusListener
ItemEvent	Генерується при виборі або скасування вибору елемента, наприклад, при клацанні на прапорці опцій	ItemListener
KeyEvent	Генерується при введенні даних з клавіатури	KeyListener
MouseEvent	Генерується при переміщенні або перетягуванні миші, натисканні чи відпусканні її клавіш, а також при приміщенні курсору миші на компонент або виведенні курсору за межі компонента	MouseListener і MouseMotionListener
MouseWheelEvent	Генерується при русі коліщатка миші	MouseWheelListener
WindowEvent	Генерується при активізації, деактивизации, закритті, згортанні і розгортанні вікна	WindowListener
	javax.swing.event	
AncestorEvent	Генерується при додаванні, переміщенні або видаленні пращура компонента	AncestorListener
CaretEvent	Генерується при зміні позиції курсора в текстовому компоненті	CaretListener
ChangeEvent	Генерується при зміні стану компонента	ChangeListener
HyperlinkEvent	Генерується при діях з гіпертекстовим посиланням	HyperlinkListener
ListDataEvent	Генерується при зміні вмісту списку	ListDataListener
ListSelectionEvent	Генерується при виборі або скасування вибору пунктів списку	ListSelectionListener
MenuEvent	Генерується при виборі або скасуванні вибору пунктів меню	MenuListener
TableModelEvent	Генерується при зміні моделі таблиці	TableModelListener
TreeExpansionEvent	Генерується при розгортанні або згортанні дерева	TreeExpansionListener
TreeModelEvent	Генерується при зміні моделі дерева	TreeModelListener
TreeSelectionEvent	Генерується при виборі вузла дерева	TreeSelectionListener

Одним з найпростіших і часто використовуваних елементів управління Swing є *Кнопка*, що представляє собою екземпляр класу JButton. Цей клас є нащадком абстрактного класу AbstractButton, в якому визначені методи, загальні для всіх компонентів, що реалізують функціонал кнопки (чек-боксів, радіокнопок), а також для меню і його елементів. На кнопці може відображатися текст та/або зображення. Клас JButton містить кілька конструкторів. Найбільш часто використовуваними є:

```
JButton(String text);
JButton(Icon icon);
JButton(String text, Icon icon);
```

Параметр `text` визначає рядок, який буде відображатися на кнопці, а параметр `icon` - зображення на ній, що завантажується з файлу, наприклад так:

```
JButton btnWithIcon = new JButton(new ImageIcon("filname.png");
```

Слід зазначити, що аналогічні конструктори має і напис JLabel.

Після клацання на кнопці генерується подія ActionEvent. Для реєстрації та відключення слухачів даної події JButton надає наступні методи (вони успадковані від класу AbstractButton):

```
void addActionListener(ActionListener al)
void removeActionListener(ActionListener al)
```

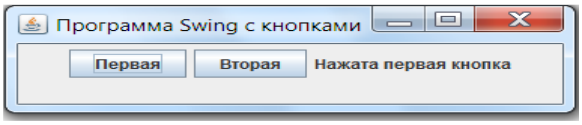
Тут параметр al задає об'єкт-слухач, який буде сповіщений про виникнення подій взаємодії користувача з кнопкою. Об'єкт повинен являти собою екземпляр класу, що реалізовує інтерфейс ActionListener. В інтерфейсі ActionListener визначений тільки один метод обробки події: `actionPerformed(ActionEvent ae)`. Даний метод викликається по клацанню на кнопці, тобто він займається обробкою подій, пов'язаних з діями користувача з кнопкою. Об'єкт `ActionEvent ae`, що передається методу `actionPerformed(ActionEvent ae)`, дозволяє отримати інформацію про компонент-джерело події, наприклад рядок *команди дії*. За умовчанням як команда дії приймається рядок, що відображається на кнопці. Для отримання команди дії треба викликати метод `getActionCommand()`, що належить об'єкту події. Він оголошується наступним чином:

```
String getActionCommand()
```

Команда дії ідентифікує кнопку. Таким чином, якщо в додатку є кілька кнопок, команда дії дозволяє досить просто визначити, яка з них стала джерелом події. Нижче наведено вихідний код і зовнішній вигляд програми, що демонструє використання двох кнопок, які реагують на дії користувача.

```
package buttonapppackage;

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

/*Пример, демонстрирующий работу с кнопками*/
public class ButtonDemo implements ActionListener {

    JLabel jlab;

    ButtonDemo() {
        JFrame jfrm = new JFrame("Программа Swing с кнопками");

        /*Установка диспетчера компоновки FlowLayout*/
        jfrm.setLayout(new FlowLayout());
        jfrm.setSize(380, 100);
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        /*Создание ДВУХ кнопок*/
        JButton jbtnFirst = new JButton("Первая");
        JButton jbtnSecond = new JButton("Вторая");
        /*Связывание с кнопками слушателя события (текущий класс
          является слушателем события, т.к. он реализует
          интерфейс ActionListener)*/
```

```
        jbtnFirst.addActionListener(this);
        jbtnSecond.addActionListener(this);
        /*Добавление кнопок на форму*/
        jfrm.add(jbtnFirst);
        jfrm.add(jbtnSecond);
        /*Создание текстовой надписи*/
        jlab = new JLabel("Нажмите кнопку");
        /*Включение надписи в состав формы*/
        jfrm.add(jlab);
        jfrm.setVisible(true);
    }

    /*Обработчик событий ActionEvent, генерируемых кнопками*/
    public void actionPerformed(ActionEvent ae) {
        /*Использование команды действия для идентификации кнопки*/
        if (ae.getActionCommand().equals("Первая")) {
            jlab.setText("Нажата первая кнопка");
        } else {
            jlab.setText("Нажата вторая кнопка");
        }
    }

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



Мал. 6. Додаток Swing, який демонструє роботу з кнопками

Розглянемо текст програми і звернемо увагу на нові для нас рішення. В першу чергу зверніть увагу на те, що в цій програмі додатково імпортуються клас java.awt.FlowLayout і класи пакету java.awt.event. Клас диспетчера компонування FlowLayout використовується для визначення принципу розміщення компонентів у фреймі (це детально буде розглянуто далі). Пакет java.awt.event необхідний, оскільки в ньому визначені інтерфейс ActionListener і клас ActionEvent.

Далі в програмі оголошується клас `ButtonDemo`, який реалізує інтерфейс `ActionListener`. Це означає, що об'єкт `ButtonDemo` використовується і як GUI-інтерфейс, і для обробки подій, відповідних дій користувача з інтерфейсом. Далі оголошується посилання на об'єкт `JLabel`. Він буде використаний в методі обробки подій `actionPerformed (ActionEvent ae)` для відображення інформації, на якій кнопці виконав клацання користувач.

Конструктор `ButtonDemo()` починається зі створення об'єкта `JFrame`, на який посилається змінна `jfrm`. Потім в конструкторі для фрейма встановлюється диспетчер компонування `FlowLayout`, який вибирає для компонентів, що розміщуються в формі, розмір, який визначається розмірами їх вмісту (в даному

випадку - розміром рядків на кнопках і в написі) (за замовчуванням з панеллю вмісту пов'язаний диспетчер компонування BorderLayout, про диспетчерів компонування буде розказано далі). `FlowLayout` розміщує компоненти "в рядок", маючи в своєму розпорядженні кожен наступний елемент праворуч від попереднього. Коли черговий рядок заповнюється, диспетчер компонування формує наступний рядок. Незважаючи на те, що дана схема забезпечує низький рівень контролю за розміщенням елементів, використовувати її дуже просто. Ви можете поекспериментувати,

Для зв'язування диспетчера компонування з панеллю вмісту форми використовується наступний оператор:

```
jfrm.setLayout(new FlowLayout());
```

До Java 5 був необхідний явний виклик методу `getContentPane()` і оператор повинен був виглядати так:

```
jfrm.getContentPane().setLayout(new FlowLayout());
```

Після установки розміру вікна і визначенні операції завершення програми при закритті вікна додатка в конструкторі `ButtonDemo()` створюються дві кнопки.

```
JButton jbtnFirst = new JButton("Первая");
JButton jbtnSecond = new JButton("Вторая");
```

На першій кнопці відображається напис `JButton jbtnFirst = new JButton("Первая");`

`JButton jbtnSecond = new JButton("Вторая");`, а на другій - Друга.

Далі до кнопок додається слухач подій, за допомогою методу `addActionListener(this)`, аргумент якого вказує на об'єкт-слухач події, в ролі якого виступає екземпляр класу `ButtonDemo` (оскільки клас реалізує інтерфейс ActionListener). Відповідні рядки коду наведені нижче.

```
jbtnFirst.addActionListener(this);
jbtnSecond.addActionListener(this);
```

В результаті виконання даних операторів об'єкт, який створив кнопки (тобто екземпляр класу `ButtonDemo`), буде отримувати сповіщення про дії з ними.

При кожному натисканні на кнопці вона генерує подію, про яку сповіщаються зареєстровані на кнопках слухачі. При цьому методу обробки подій слухача `actionPerformed(ActionEvent ae)` передається як параметр об'єкт-подія `ActionEvent`, що представляє подію взаємодії користувача з кнопкою. У програмі `ButtonDemo` метод обробки подій має такий вигляд:

```
public void actionPerformed(ActionEvent ae) {
    if (ae.getActionCommand().equals("Первая")) {
        jlab.setText("Нажата первая кнопка");
    } else {
        jlab.setText("Нажата вторая кнопка");
    }    }
```

У тілі методу з об'єкта-події `ae` методом `getActionCommand()` витягується команда дії, яка відповідає напису на кнопці, що згенерувала подію. Залежно від вмісту рядка, що представляє команду дії, встановлюється текст напису.

Розташування компонентів в складі контейнера визначається *диспетчером компонування* (*Layout Manager*), пов'язаним з цим контейнером. Диспетчери компонування визначають розмір і розташування компонентів, а при зміні розміру контейнера пропорційно масштабують компоненти. В Java визначено кілька таких диспетчерів. Більшість з них входять до складу AWT (пакет java.awt), але кілька додаткових диспетчерів компонування надає Swing. Всі диспетчери є екземплярами класів, що реалізують інтерфейс java.awt.LayoutManager (деякі з диспетчерів реалізують інтерфейс java.awt.LayoutManager2). Нижче описані кілька популярних диспетчерів компонування, доступних програмістам, які використовують засоби AWT та Swing.

Табл. 2. Популярні диспетчери компонування AWT та Swing

FlowLayout	Розміщує компоненти в рядку зліва направо; наступний рядок розміщується під попереднім (налаштування для деяких країн передбачають розміщення компонентів в рядку справа наліво). Цей диспетчер компонування пов'язаний з контейнером JPanel за замовчуванням
BorderLayout	Поміщає компоненти в п'яти областях, розташованих по центру і по краях контейнера, "розтягуючи" їх на всю область. Цей диспетчер компонування пов'язаний з панеллю вмісту контейнера JFrame за замовчуванням
GridLayout	Розміщує компоненти у вигляді таблиці з вказаною кількістю рядків і стовпців і інтервалами між ними. Розтягує розмір компонентів на всю область комірки
GridBagLayout	Розміщує компоненти у вигляді таблиці з комірками різних розмірів
BoxLayout	Розміщує компоненти у вертикальній або горизонтальній смузі
SpringLayout	Використовує при розміщенні компонентів спеціальні обмеження
GroupLayout	Розкладає компоненти по групах. Групи мають напрямок по осі і можуть бути *паралельними* і *послідовними*. У послідовній групі у кожного наступного компонента координата уздовж осі на одиницю більше (мається на увазі координата в сітці), в паралельній - компоненти мають одну і ту ж координату. Використовується за замовчуванням візуальним редактором графічного інтерфейсу IDE NetBeans

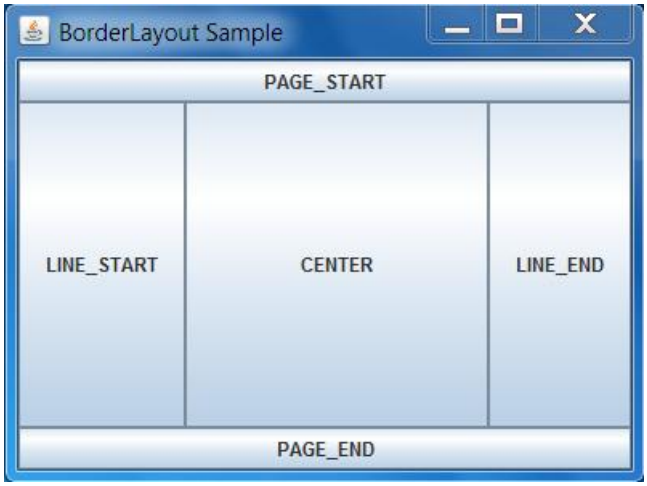
До сих пір нам зустрічалися два диспетчера: BorderLayout (який за замовчуванням пов'язаний з панеллю вмісту JFrame) і FlowLayout (є диспетчером компонування за замовчуванням контейнера JPanel).

Диспетчер BorderLayout визначає в складі контейнера п'ять областей, в яких можуть розміщатись компоненти. Перша область розташована по центру вікна, решта чотири - по краях. Відповідно, області називаються *центр* (*Center*), *північ* (*North*), *південь* (*South*), *схід* (*East*) та *захід* (*West*). За замовчуванням компонент, який розміщується в панель вмісту, розташовується в центральній області. Явно управляти розміщенням компонентів можна, використовуючи спеціальну форму методу додавання компонента:

```
void add(Component comp, Object loc);
```

де параметр `comp` задає компонент, що додається до панелі, а параметр `loc` визначає область, в якій цей компонент буде розміщений. Допустимі такі значення параметра `loc` (Мал. 7):

Верхня частина вікна	Права частина вікна	Нижня частина вікна	Ліва частина вікна	Центральна частина вікна
`BorderLayout .PAGE_START`	`BorderLayout .LINE_END`	`BorderLayout .PAGE_END`	`BorderLayout .LINE_START`	`BorderLayout .CENTER`
`BorderLayout .NORTH "North"`	`BorderLayout .EAST "East"`	`BorderLayout .SOUTH "South"`	`BorderLayout .WEST "West"`	`BorderLayout .CENTER "Center"`



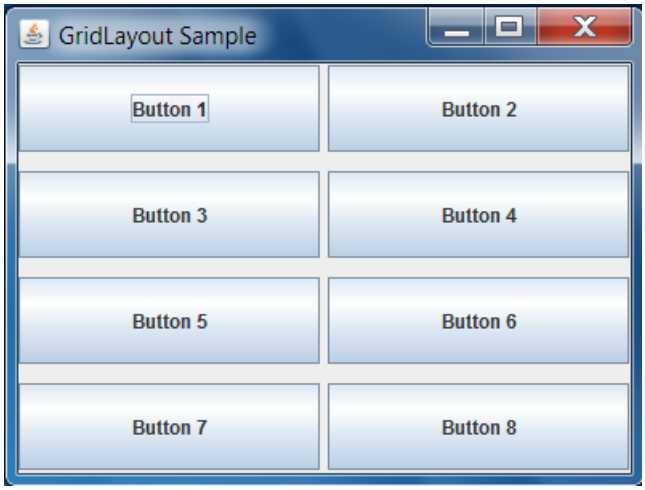
Мал. 7. Додаток з кнопками, розміщеними в областях диспетчера компонування BorderLayout

Диспетчер компонування BorderLayout найбільш зручний, якщо ви створюєте об'єкт JFrame, який повинен містити лише один компонент (він розміщується в центрі вікна), або якщо вам необхідно розмістити в різних областях об'єкти-панелі, що угруповують візуальні компоненти.

При необхідності розташування компонентів на перетині рядків і стовпців таблиці (Мал. 8) менеджером компонування контейнера встановлюється об'єкт GridLayout, який створюють, наприклад, так:

```
GridLayout gl = new GridLayout(4, 0, 5, 12);
```

Перший параметр конструктора передає бажану кількість рядків, другий - кількість стовпців (якщо кількість рядків або кількість стовпців встановити в 0, то вони обчислюються динамічно при додаванні компонентів, але один з цих параметрів повинен бути вказаний явно). Третій і четвертий параметр задають інтервали в пікселах між рядками і стовпцями, відповідно.

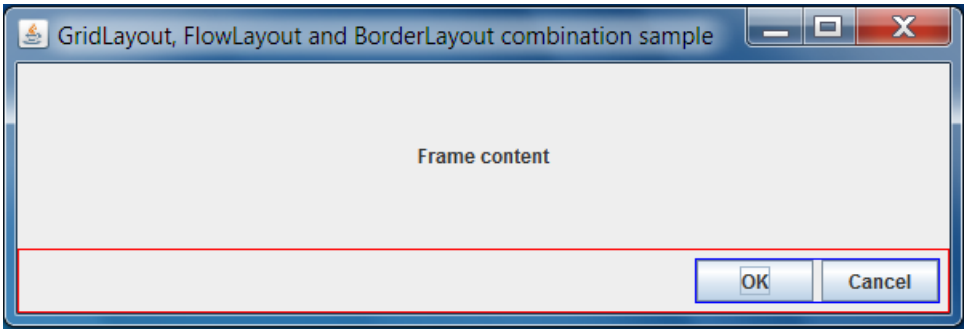


Мал. 8. Додаток з кнопками, розміщеними з використанням диспетчера компонування GridLayout

Нижче наводиться текст програми, яка використовує комбінацію диспетчерів компонування, і зовнішній вигляд відповідного додатку (Мал. 9):

```
package layouts;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import javax.swing.*;

/**
 * Пример совместного использования менеджеров компоновки.
*
 */
public class CompositeLayoutSample extends JFrame {

    public CompositeLayoutSample() {
        super("GridLayout, FlowLayout and BorderLayout combination "
            + "sample");
        setSize(600, 200);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        /*Используем для панели кнопок GridLayout (1 ряд 2 столбца)
          с целью обеспечения одинаковых размеров кнопок*/
        JPanel grid = new JPanel(new GridLayout(1, 2, 5, 0));
        /*Обрамим панель кнопок рамкой синего цвета*/
        grid.setBorder(BorderFactory.createLineBorder(Color.blue));
        grid.add(new JButton("OK"));
        grid.add(new JButton("Cancel"));

        /*Панель, содержащая панель кнопок, использует FlowLayout
          для обеспечения постоянных размеров кнопок (равного размеру
          максимальной) независимо от размера формы.
          Панель кнопок выравнивается по правому краю*/
        JPanel flow = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        /*Обрамим панель, содержащую панель кнопок, рамкой
          красного цвета*/
        flow.setBorder(BorderFactory.createLineBorder(Color.red));
```

```
            flow.add(grid);
            /*Панель содержимого формы использует BorderLayout,
              размещаем в нижней её части панель flow,
              содержащую панель кнопок*/
            add(flow, BorderLayout.PAGE_END);
            /*Для заполнения основного пространства формы добавим
              надпись, конструктор надписи содержит параметр выравнивания
              по центру по горизонтали, выравнивание по центру
              по вертикали обеспечивает BorderLayout*/
            add(new JLabel("Frame content", JLabel.CENTER));
            /*Выравниваем форму относительно центра экрана*/
            setLocationRelativeTo(null);
            setVisible(true);
        }

        public static void main(String[] args) {
            CompositeLayoutSample app = new CompositeLayoutSample();
        }
    }
```



Мал. 9. Додаток, що використовує комбінацію диспетчерів компонування

Звернемо увагу, що панель з диспетчером компонування GridLayout, на відміну від BoxLayout, забезпечує однаковий розмір кнопок (визначається розміром напису максимального розміру на кнопці). Також продемонстровано використання конструктора диспетчера компонування FlowLayout з параметром, що визначає вирівнювання вмісту панелі:

```
new FlowLayout(FlowLayout.RIGHT);
```

Для виділення панелей на формі використовується обрамлення панелей лініями різних кольорів із задіянням функціоналу класів javax.swing.BorderFactory і java.awt.Color:

```
grid.setBorder(BorderFactory.createLineBorder(Color.blue));
```

Результатом використання комбінації диспетчерів компонування є зовнішній вигляд інтерфейсу, у якого кнопки управління "прив'язані" до нижньої правої частини вікна і не змінюють свого розміру і взаємного розташування при змінах розміру форми.

Зверніть увагу на оператор

```
setLocationRelativeTo(null);
```

який дозволяє вирівнювати вікно форми додатка по центру екрану під час запуску програми.

Крім написів і кнопок до числа широко використовуваних компонентів відноситься текстове поле JTextField. Цей компонент дозволяє користувачеві вводити рядок тексту. JTextField є підкласом абстрактного класу javax.swing.text.JTextComponent, який виступає в ролі суперкласу не тільки для JTextField, але і для інших текстових компонентів.

У класі JTextField визначено кілька конструкторів. Один з них виглядає наступним чином:

```
JTextField(int cols)
```

де параметр `cols` визначає ширину текстового поля, виражену в кількості позицій для введення символів. Відзначимо, що довжина рядка, що вводиться, не обмежується шириною поля, яка відображається на екрані.

Закінчивши введення тексту в поле, користувач натискає клавішу *Enter*, в результаті чого компонент JTextField генерує подію ActionEvent. Клас JTextField надає розробнику методи `addActionListener(ActionListener al)` і `removeActionListener(ActionListener al)`. Для обробки подій необхідно реалізувати метод `actionPerformed(ActionEvent ae)`, оголошений в інтерфейсі ActionListener. Обробка подій текстового поля здійснюється так само, як і обробка подій кнопки, про яку йшла мова раніше.

Щоб отримати рядок, що відображається в компоненті, треба з об'єкта JTextField викликати метод `getText()`. Оголошення цього методу наведено нижче.

```
String getText ();
```

Задати текст для компонента JTextField дозволяє метод `setText`, який має наступний вигляд:

```
void setText(String text)
```

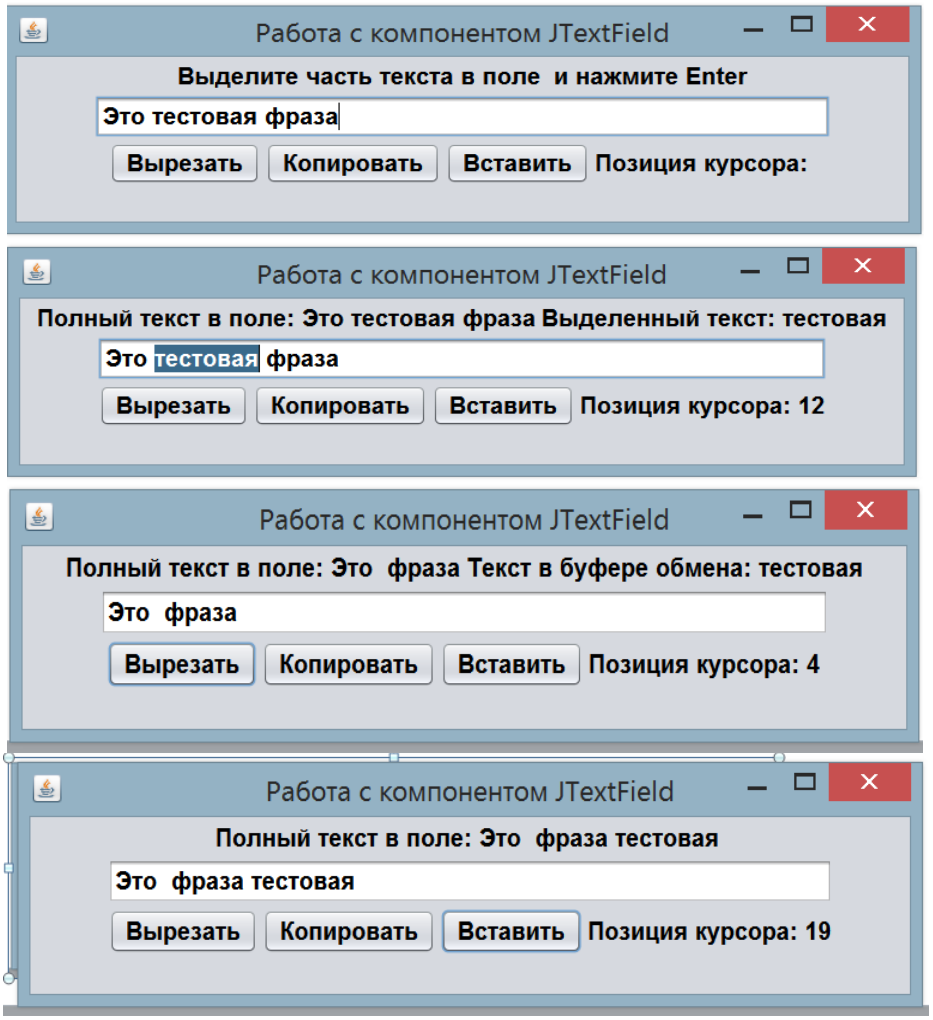
Рядок тексту передається компоненту за допомогою параметра `text`.

Далі наведено вихідний код і зовнішній вигляд програми, що демонструє використання компонента JTextField (Мал. 10). Додаток показує можливість роботи з Буфером обміну операційної системи за допомогою операцій вирізання, копіювання і вставки рядків. Зверніть увагу на те, що клас TextFieldDemo успадковує клас JFrame, що робить непотрібним створення форми в конструкторі (тобто клас TextFieldDemo - це форма).

```
package textfieldapppackage;


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

/*Демонстрация работы с текстовым полем*/
public class TextFieldDemo extends JFrame {
    JLabel jlabAll;        //отображает всю строку
    JLabel jlabSelected;   //отображает выделенную часть строки
```

```
JLabel jlabCurPos;     //отображает позицию курсора
JTextField jtf;
JButton jbtnCut;
JButton jbtnCopy;
JButton jbtnPaste;

public TextFieldDemo() {
    setTitle("Работа с компонентом JTextField");
    setLayout(new FlowLayout());
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(400, 200);

    jlabAll = new JLabel("Выделите часть текста в поле ");
    jlabSelected = new JLabel("и нажмите Enter");
    add(jlabAll);
    add(jlabSelected);
    /*Создаётся текстовое поле с первоначальной фразой
      и предпочитаемой шириной*/
    jtf = new JTextField("Это тестовая фраза", 35);
    add(jtf);
    /*Добавление слушателя события - нажатия клавиши Enter
      организуется через создание объекта анонимного класса,
      реализующего интерфейс ActionListener */
    jtf.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jlabAll.setText("Полный текст в поле: "
                            + jtf.getText());
            jlabSelected.setText("Выделенный текст: "
                            + jtf.getSelectedText());
        }
    });
    /*Добавление кнопок*/
    jbtnCut = new JButton("Вырезать");
    jbtnCopy = new JButton("Копировать");
    jbtnPaste = new JButton("Вставить");
    add(jbtnCut);
    add(jbtnCopy);
    add(jbtnPaste);
    /*Добавление слушателя события - нажатия кнопки Вырезать*/
    jbtnCut.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {
            /*Вырезание выделенного текста в буфер обмена*/
            jtf.cut();
            /*Отображение оставшейся части текста*/
            jlabAll.setText("Полный текст в поле: "
                            + jtf.getText());
            /*Создание вспомогательного текстового поля,
              куда копируется текст из буфера обмена
              для вывода его в метке*/
            JTextField buftf = new JTextField();
            buftf.paste();
            jlabSelected.setText("Текст в буфере обмена: "
                                 + buftf.getText());
        }
```

```
        });
        /*Добавление слушателя события нажатия на кнопку Копировать*/
        jbtnCopy.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                /*Копирование выделенного текста в буфер обмена*/
                jtf.copy();
                /*Создание вспомогательного текстового поля,
                  куда копируется текст из буфера обмена
                  для вывода его в метке*/
                JTextField buftf = new JTextField();
                buftf.paste();
                jlabSelected.setText("Текст в буфере обмена: "
                                     + buftf.getText());
            }
        });
        /*Добавление слушателя события нажатия на кнопку Вставить.
          Перед нажатием на кнопку Вставить можно менять позицию
          курсора, указывая место вставки*/
        jbtnPaste.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                jtf.paste();
                jlabAll.setText("Полный текст в поле: "
                                + jtf.getText());
                jlabSelected.setText("");
            }
        });
        /*Добавление слушателя, отслеживающего события
          изменения позиции курсора*/
        jlabCurPos = new JLabel("Позиция курсора: ");
        jtf.addCaretListener(new CaretListener() {

            @Override
            public void caretUpdate(CaretEvent e) {
                jlabCurPos.setText("Позиция курсора: "
                                   + jtf.getCaretPosition());
            }
        });
        add(jlabCurPos);
        setVisible(true);
    }

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

            @Override
            public void run() {
                TextFieldDemo tfd = new TextFieldDemo();
            }
        });
    }
}
```



Мал. 10. Додаток, яке демонструє роботу з текстовим полем

Операції роботи з Буфером обміну файлової системи забезпечують методи абстрактного класу javax.swing.text.JTextComponent `void copy()`, `void cut()` і `void paste()`. У програмі вміст Буфера обміну дублюється в додатковому текстовому полі для можливості його передачі в напис.

У наведеній вище програмі до текстового поля додається слухач javax.swing.event.CaretListener, що відслідковує зміну позиції курсора в текстовому полі.

Подібно JButton, з компонентом JTextField пов'язана команда дії. За замовченням як рядок такої команди приймається поточний вміст поля редагування. Однак Ви можете встановити команду явним чином, викликавши метод `setActionCommand`.

```
void setActionCommand(String cmd)
```

Рядок, переданий за допомогою параметра `cmd`, стає новою командою дії; при цьому текст в текстовому полі не змінюється. Як правило, розробники вдаються до явного налаштування команди дії для того, щоб забезпечити розпізнавання компонента, який згенерував подію. Так доводиться поступати у тому випадку, якщо в формі знаходиться декілька елементів управління, для

яких розроблений загальний обробник подій. Встановивши команду дії, Ви отримуєте зручний засіб ідентифікації компонента.

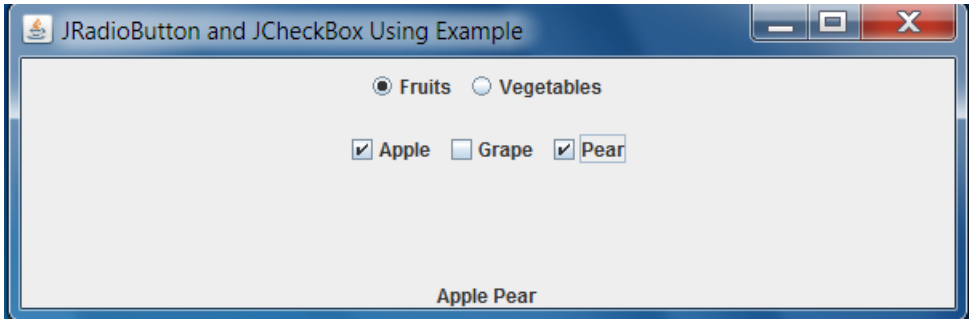
Іншими спадкоємцями абстрактного класу javax.swing.text.JTextComponent є JTextArea і JEditorPane. Обидва дозволяють відображати багаторядковий текст, але перший - написаний тільки одним шрифтом, а другий - підтримує форматування шрифтів і вставку зображень.

Популярними компонентами також є JCheckBox (прапорець) і JRadioButton (перемикач). Для забезпечення функціональності останнього кілька примірників JRadioButton зазвичай об'єднуються в групу - об'єкт ButtonGroup. Нижче наводиться текст і звонішній вигляд програми, що демонструє використання зазначених компонентів:

```
import java.awt.BorderLayout;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import javax.swing.*;

/**
 * Демонстрация использования компонентов JRadioButton и JCheckBox,
 * а также динамической замены контейнеров JPanel.
 */
public class ComponentRadioButtonCheckBox extends JFrame {

    private final JPanel pnlChoose;
    private final JPanel pnlContent;
    private final JPanel pnlFruitContent;
    private final JPanel pnlVegetableContent;
    private final JCheckBox chkApple;
    private final JCheckBox chkGrape;
    private final JCheckBox chkPear;
    private final JCheckBox chkTomato;
    private final JCheckBox chkCucumber;
    private final JCheckBox chkPotato;
    private final JLabel lblInfo;


    public ComponentRadioButtonCheckBox() {
        super("JRadioButton and JCheckBox Using Example");
        setSize(600, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        /*Панель выбора фрукты или овощи - группа радиокнопок*/
        pnlChoose = new JPanel();
        JRadioButton rbtnFruits = new JRadioButton("Fruits");
        JRadioButton rbtnVegetables = new JRadioButton("Vegetables");
        ButtonGroup btngrChoose = new ButtonGroup();
        btngrChoose.add(rbtnFruits);
        btngrChoose.add(rbtnVegetables);
        pnlChoose.add(rbtnFruits);
        pnlChoose.add(rbtnVegetables);
        add(pnlChoose, BorderLayout.NORTH);
        pnlContent = new JPanel();
        add(pnlContent, BorderLayout.CENTER);
        /*Панель выбора фруктов - чекбоксы*/
        pnlFruitContent = new JPanel();
```

```
        chkApple = new JCheckBox("Apple");
        chkGrape = new JCheckBox("Grape");
        chkPear = new JCheckBox("Pear");
        pnlFruitContent.add(chkApple);
        pnlFruitContent.add(chkGrape);
        pnlFruitContent.add(chkPear);
        /*Панель выбора овощей - чекбоксы*/
        pnlVegetableContent = new JPanel();
        chkTomato = new JCheckBox("Tomato");
        chkCucumber = new JCheckBox("Cucumber");
        chkPotato = new JCheckBox("Potato");
        pnlVegetableContent.add(chkTomato);
        pnlVegetableContent.add(chkCucumber);
        pnlVegetableContent.add(chkPotato);
        /*Информационная строка - отображает список выбранных фруктов
          или овощей*/
        lblInfo = new JLabel("", JLabel.CENTER);
        add(lblInfo, BorderLayout.SOUTH);

        MyRadioButtonsListener radioButtonsListener =
                                        new MyRadioButtonsListener();
        rbtnFruits.addItemListener(radioButtonsListener);
        rbtnVegetables.addItemListener(radioButtonsListener);

        MyCheckBoxListener fruitsCheckBoxListener =
                 new MyCheckBoxListener(chkApple, chkGrape, chkPear);
        chkApple.addItemListener(fruitsCheckBoxListener);
        chkGrape.addItemListener(fruitsCheckBoxListener);
        chkPear.addItemListener(fruitsCheckBoxListener);

        MyCheckBoxListener vegetablesCheckBoxListener =
           new MyCheckBoxListener(chkTomato, chkCucumber, chkPotato);
        chkTomato.addItemListener(vegetablesCheckBoxListener);
        chkCucumber.addItemListener(vegetablesCheckBoxListener);
        chkPotato.addItemListener(vegetablesCheckBoxListener);

        setVisible(true);
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new ComponentRadioButtonCheckBox();
            }
        });
    }

    /**
     * Слушатель выбора радиокнопок фрукты или овощи.
     */
    private class MyRadioButtonsListener implements ItemListener {

        @Override
        public void itemStateChanged(ItemEvent e) {
            /*Очистка*/
            pnlContent.removeAll();
```

```
            lblInfo.setText("");
            chkApple.setSelected(false);
            chkGrape.setSelected(false);
            chkPear.setSelected(false);
            chkTomato.setSelected(false);
            chkCucumber.setSelected(false);
            chkPotato.setSelected(false);
            /*Перерисовка компонента - указывает Swing, что нужно
              удалить компоненты, удалённые методом remove() или
              removeAll()*/
            pnlContent.repaint();
            /*Указывает менеджеру компоновки на необходимость
              пересчёта компоновки, что необходимо при добавлении
              новых компонентов*/
            pnlContent.revalidate();
            /*Установка панели, соответствующей выбранной
              радиокнопке*/
            if ("Fruits".equals(((JRadioButton) e.getSource())
                                                .getText())) {
                pnlContent.add(pnlFruitContent, BorderLayout.CENTER);
            } else {
                pnlContent.add(pnlVegetableContent,
                               BorderLayout.CENTER);
            }
            pnlContent.repaint();
            pnlContent.revalidate();
        }
    }

    /**
     * Слушатель выбора овощей или фруктов.
     */
    private class MyCheckBoxListener implements ItemListener {
        private final JCheckBox[] items;

        public MyCheckBoxListener(JCheckBox… items) {
            this.items = items;
        }
        @Override
        public void itemStateChanged(ItemEvent e) {
            StringBuilder selected = new StringBuilder();
            for (JCheckBox item : items) {
                if (item.isSelected()) {
                    selected.append(item.getText()).append(" ");
                }
            }
            lblInfo.setText(selected.toString());
        }
    }
}
```



Мал. 11. Додаток, яке демонструє роботу з перемикачами і прапорцями

Зверніть увагу на використання в програмі для угруповання компонентів об'єктів JPanel, а також організовану в класі слухача MyRadioButtonsListener динамічну заміну і відображення об'єктів JPanel в залежності від обраної користувачем радіокнопки.

Swing також підтримує компоненти вибору зі списку елементів, такі як JList і JComboBox. Наведена нижче програма демонструє використання компонента JList:

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

/**
 * Класс, демонстрирующий работу с компонентом JList.
 */
public class ComponentList {

    public ComponentList() {
        final JFrame jfrm = new JFrame("Компонент JList");
        jfrm.setLayout(new FlowLayout());
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jfrm.setSize(500, 300);
        final JLabel jlab = new JLabel();
        jfrm.add(jlab);
        /*Массив с данными списка (вместо массива также могут
          использоваться вектор и модель списка)*/
        final String[] data = {"one", "two", "three", "four", "five",
                               "six", "seven", "eight", "nine", "ten"};
        /*При создании списка передаём в конструктор массив строк*/
        final JList datalist = new JList(data);
        /*Установка режима, позволяющего выбирать только один элемент
          списка, ещё варианты SINGLE_INTERVAL_SELECTION
          и MULTIPLE_INTERVAL_SELECTION*/
        datalist.setSelectionMode(ListSelectionModel
                                  .SINGLE_SELECTION);
        /*Для отображения списка в прокручиваемом окне создаём объект
          JScrollPane, в конструктор которого передаём созданный
          список*/
        JScrollPane jscpane = new JScrollPane(datalist);
        /*Установка размера прокручиваемой панели*/
        jscpane.setPreferredSize(new Dimension(120, 160));
        jfrm.add(jscpane);
```

```
        /*Создаём слушатель события выбора элемента в списке
          и реализуем абстрактный метод valueChanged класса
          ListSelectionListener, получающий событие
          ListSelectionEvent*/
        ListSelectionListener lsldatalist =
                                        new ListSelectionListener() {
            @Override
            public void valueChanged(ListSelectionEvent e) {
                /*Получаем индекс выбранного элемента, если ни один
                  элемент не выбран возвращается -1*/
                int i = datalist.getSelectedIndex();
                if (i != -1) {
                    jlab.setText("You choose " + data[i]);
                } else {
                    jlab.setText("You don't choose list item");
                }
            }
        };
        datalist.addListSelectionListener(lsldatalist);
        jfrm.setVisible(true);
    }

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

            @Override
            public void run() {
                ComponentList comlst = new ComponentList();
            }
        });
    }
}
```

Конструктор об'єктів JList приймає як параметр масив рядків, існують перевантажені конструктори, які приймають колекцію java.util.Vector або об'єкт класу, що реалізує інтерфейс javax.swing.ListModel (часто javax.swing.DefaultListModel, який має методи додання, отримання, видалення та пошуку елементів списку) в якості параметра.

Для списку існує можливість завдання режиму вибору елементів списку шляхом передачі в метод `void setSelectionMode (int selectionMode)` об'єкта JList константи `ListSelectionModel.SINGLE_SELECTION` - для режиму вибору тільки одного елемента зі списку, `ListSelectionModel.SINGLE_INTERVAL_SELECTION` - для режиму вибору одного діапазону розташованих підряд елементів списку і `ListSelectionModel.MULTIPLE_INTERVAL_SELECTION` - для вибору декількох діапазонів елементів списку.

З огляду на те, що список може не вміщатися у відведену йому область, часто список обертають в компонент JScrollPane (*Панель з прокруткою*). Розмір такої панелі встановлюється методом `void setPreferredSize(Dimension`

preferredSize), який отримує в якості параметра об'єкт java.awt.Dimension, який інкапсулює ширину і висоту області, відведеної для панелі з прокруткою.

Вибір елементів в списку відстежується слухачем подій javax.swing.event.ListSelectionListener з методом обробки події void valueChanged(ListSelectionEvent e). У реалізації цього методу зазвичай отримують індекс елемента списку (методом getSelectedIndex() класу JList), за допомогою якого витягують і використовують відповідні дані з масиву, переданого конструктору списку.

Компонент JComboBox відрізняється від JList можливістю організації редагування вибраного елементу списку. Також, на відміну від JList, об'єкт JComboBox має тільки один режим вибору - дозволяє вибирати тільки один елемент. Наведений нижче код демонструє роботу компонента JComboBox. Зверніть увагу на те, що клас ComponentComboBox є одночасно і формою, і слухачем подій.

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

/**
 * Демонстрация работы компонента JComboBox.
 */
public class ComponentComboBox extends JFrame
                                        implements ActionListener {

    private final JLabel jlab;

    public ComponentComboBox() {
        super("Компонент JComboBox");
        setSize(600, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel pnlRoot = new JPanel();
        /*Массив с данными списка (вместо массива также могут
          использоваться вектор и модель списка)*/
        final String[] data = {"one", "two", "three", "four", "five",
                               "six", "seven", "eight", "nine", "ten"};
        JComboBox comboBox = new JComboBox(data);
        /*Делаем компонент редактируемым*/
        comboBox.setEditable(true);
        comboBox.addActionListener(this);
        JScrollPane scrollPane = new JScrollPane(comboBox);
        pnlRoot.add(scrollPane);
        add(pnlRoot, BorderLayout.NORTH);
        jlab = new JLabel("", JLabel.CENTER);
        add(jlab, BorderLayout.SOUTH);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        JComboBox jcb = (JComboBox) e.getSource();
        String item = (String) jcb.getSelectedItem();
        jlab.setText("Ваш выбор: " + item);
```

```
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new ComponentComboBox();
            }
        });
    }
}
```

У обробнику події продемонстровано отримання з події методом `Object getSource()` посилання на компонент-джерело події.

Для віконних додатків часто необхідно створювати меню команд. Swing підтримує створення меню за допомогою декількох компонентів: `JMenuBar` (Рядок меню), `JMenu` (Меню команди), `JMenuItem` (Пункт меню). Наступний приклад демонструє побудову і організацію роботи з меню:

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.io.*;
import javax.swing.*;
import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;

/**
 * Демонстрация построения и работы с меню.
*/
public class SimpleMenu extends JFrame {

    static JTextArea info = new JTextArea(500, 300);

    public SimpleMenu () {
        setTitle("Работа с меню и его элементами");
        setSize(600, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(info, BorderLayout.CENTER);
        JMenuBar menuBar = new JMenuBar();
        JMenu file = new JMenu("File");
        file.setMnemonic(KeyEvent.VK_F);
        JMenuItem exit = new JMenuItem("Exit");
        exit.setMnemonic(KeyEvent.VK_X);
        exit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
        JMenuItem open = new JMenuItem("Open");
        open.setMnemonic(KeyEvent.VK_O);
        open.addActionListener(new OpenActionListener());
```

```
        JMenuItem saveAs = new JMenuItem("Save As");
        saveAs.setMnemonic(KeyEvent.VK_S);
        saveAs.addActionListener(new MenuActionListener(info));

        file.add(open);
        file.add(saveAs);
        file.addSeparator();
        file.add(exit);
        menuBar.add(file);
        /*Устанавливаем панель меню в форму*/
        setJMenuBar(menuBar);
        setVisible(true);
    }

    private static class OpenActionListener implements ActionListener
                                                                    {
        @Override
        public void actionPerformed(ActionEvent e) {
            File currentDirectory =
                            new File(System.getProperty("user.dir"));
            JFileChooser fileChooser =
                                    new JFileChooser(currentDirectory);
            FileFilter filter =
                    new FileNameExtensionFilter("Text files", "txt");
            fileChooser.setFileFilter(filter);
            int result = fileChooser
                           .showOpenDialog((JMenuItem) e.getSource());
            if (result == JFileChooser.APPROVE_OPTION) {
                File selectedFile = fileChooser.getSelectedFile();
                try (BufferedReader br = new BufferedReader(new
InputStreamReader(new FileInputStream(selectedFile), "CP1251"))) {
                    StringBuilder sb = new StringBuilder();
                    String s;
                    while ((s = br.readLine()) != null) {
                        sb.append(s+"\n");
                    }
                    info.setText(sb.toString());
                } catch (IOException ex) {
                    System.out.println(ex.getMessage());
                }
            }
        }
    }
    private class MenuActionListener implements ActionListener {

        JTextArea info;

        public MenuActionListener(JTextArea info) {
            this.info = info;
        }

        @Override
        public void actionPerformed(ActionEvent e) {
            String item = e.getActionCommand();
            info.setText(item);
        }
```

```
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new SimpleMenu();
            }
        });
    }
}
```

За допомогою методу `void setMnemonic(int mnemonic)` для об'єктів JMenuItem можна задавати клавіатурні комбінації виклику, використовуючи як аргумент відповідну обраній клавіші (використовується разом з *Alt*) константу класу java.awt.event.KeyEvent. До об'єкту JMenu, крім елементів меню, за допомогою виклику методу `void addSeparator()` може бути доданий роздільник. У обробнику подій MenuActionListener текст елемента меню отримується із об'єкта події методом `getActionCommand()` і додається до компоненту JTextArea (Текстова область).

Для елемента меню відкриття файлу використовується окремий обробник OpenActionListener, що використовує компонент JFileChooser для вибору файлу, вміст якого відображається в області тексту. Для зменшення операцій переходу між каталогами в конструктор цього об'єкту можна передати об'єкт File, який вказує на каталог з якого запускається додаток, що отримується методом `getProperty("user.dir")` класу java.lang.System. У наведеному нижче прикладі список відображуваних об'єктом JFileChooser файлів обмежується об'єктом FileFilter. У Swing існує реалізація цього інтерфейсу - javax.swing.filechooser.FileNameExtensionFilter, що дозволяє побудувати фільтр по одному або декільком розширень імен файлів. Вікно вибору файлів відкривається викликом з об'єкта JFileChooser методу `showOpenDialog((JMenuItem) e.getSource())`, якому передається посилання на джерело події вибору даного пункту меню. Відзначимо, що в класі JFileChooser також існують методи:

```
int showSaveDialog(Component parent)
int showDialog(Component parent, String approveButtonText)
```

Перший відкриває вікно збереження файлу, а другий - вікно з кнопкою з написом-рядком, який передається другим параметром (наприклад Run). Усі три методи повертають код виконання операції, який пов'язаний з константами:

`JFileChooser.APPROVE_OPTION` - у разі успішного вибору файлу

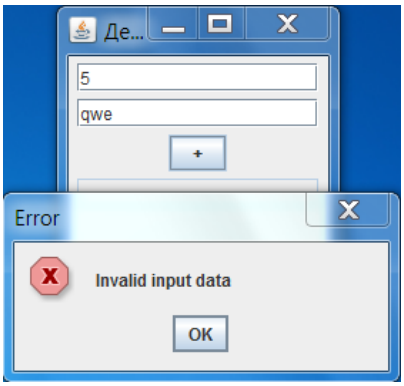
`JFileChooser.CANCEL_OPTION` - у разі скасування вибору

`JFileChooser.ERROR_OPTION` - у разі помилки

Обраний файл у вигляді об'єкта File повертається об'єктом JFileChooser при виклику з нього методу `getSelectedFile()`. Відзначимо, що з об'єкта JFileChooser також може бути викликаний метод `File getCurrentDirectory()`, який повертає у вигляді об'єкта File обраний каталог.

У додатках часто необхідно повідомити користувачеві про якісь проблеми або зажадати від нього додаткові дані. Це можна виконати за допомогою модального і немодального діалогового вікна - екземпляра класу JDialog. Оскільки частіше використовуються модальні діалогові вікна, в Swing доданий відповідний клас javax.swing.JOptionPane з багатим функціоналом. Нижче наведений приклад коду метода обробки події натискання на кнопку, який складає два числа, отриманих з текстових полів, і вид додатка з повідомленням про помилку (Мал. 12):

```
public void actionPerformed(ActionEvent e) {
    try {
        double num1 = Double.parseDouble(tfNum1.getText());
        double num2 = Double.parseDouble(tfNum2.getText());
        tfRes.setText(String.valueOf(num1 + num2));
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(this, "Invalid input data",
                                      "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```



Мал. 12. Додаток, який демонструє роботу модального діалогового вікна з повідомленням про помилку

В разі. якщо вводяться не числа або числа, що перевищують допустимий діапазон, генерується виключення NumberFormatException, в блоці перехоплення якого викликається модальное діалогове вікно статичним методом `void showMessageDialog(Component parentComponent, Object message, String title, int messageType)`. Параметр `parentComponent` вказує батьківський для діалогового вікна контейнер, `message` - відображається як текст напису у вікні, `title` - текст заголовка, а `messageType` - передає константу, що визначає іконку, яка додається у вікно. Доступні такі іконки:

 - `JOptionPane.QUESTION_MESSAGE`

 - `JOptionPane.INFORMATION_MESSAGE`

 - `JOptionPane.WARNING_MESSAGE`

 - `JOptionPane.ERROR_MESSAGE`

без іконки - `JOptionPane.PLAIN_MESSAGE`

З об'єкта JOptionPane альтернативно можуть бути викликані статичні методи (показані їх перевантажені версії з максимальною кількістю параметрів):

```
int showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon)
int showOptionDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)
int showInternalConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon)
Object showInputDialog(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)
Object showInternalInputDialog(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)
```

Вони дозволяють за допомогою параметра `int optionType` керувати розміщенням кнопок через константи:

```
JOptionPane.DEFAULT_OPTION
JOptionPane.YES_NO_OPTION
JOptionPane.YES_NO_CANCEL_OPTION
JOptionPane.OK_CANCEL_OPTION
```

Два останніх методи дозволяють розміщувати компоненти, що підтримують введення даних користувачем. Параметр `Icon icon` дозволяє замінювати іконку за замовчуванням у вікні на користувацьку (це можливо і для методу `showMessageDialog`).

Слід зазначити, що крім описаних вище компонентів в бібліотеці Swing присутні і інші, як прості, такі як JPasswordField, JToggleButton, JProgressBar та ін., так і більш складні, засновані на використанні моделей - JTable і JTree та ін. Вивчити роботу з ними рекомендується за допомогою літератури, наведеної в кінці даної лабораторної роботи.

2. Виконання роботи

GUI Builder IDE NetBeans дозволяє візуально створювати додатки на мові Java з графічним інтерфейсом з використанням компонентів Swing. Розглянемо розробку проекту графічного інтерфейсу з додаванням ряду кнопок і текстових полів. Текстові поля призначені для отримання даних, які вводяться користувачем, і виведення результату роботи програми. Кнопка ініціюватиме роботу функцій, вбудованих в клієнтську частину програми. Створюваний додаток являє собою простий калькулятор.

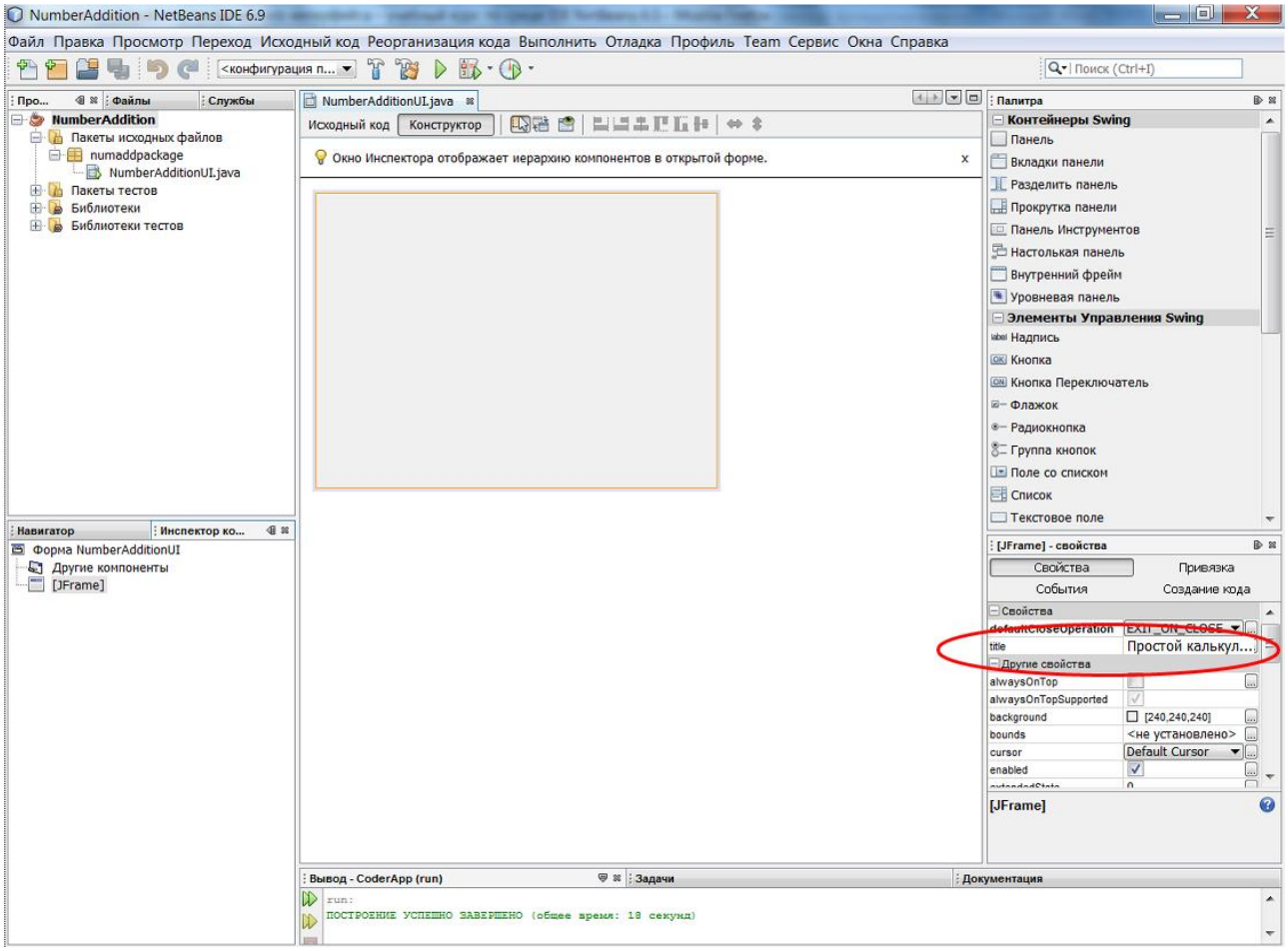
Першою дією є створення проекту у IDE. Дамо проекту ім'я NumberAddition. Виберіть *Файл-Створити проект*, в області *Категорії* виберіть вузол *Java*. В області *Проекти* виберіть *Додаток Java*. Натисніть кнопку *Далі*. Введіть NumberAddition в поле *Ім'я проекту* і вкажіть шлях до цього проекту, наприклад, в домашньому каталозі. Якщо встановлений прапорець *Зробити головним проектом*, видаліть прапорець. Натисніть кнопку *Завершити*. Оскільки не реко-

мендується створювати класи в пакеті за замовчуванням у вікні *Проекти* клацніть на вузлі *Пакети вихідних файлів* правою кнопкою і виберіть команду *Створити-Пакет Java*. Введіть ім'я `numaddpackage` в поле *Ім'я пакета*, потім натисніть Завершити. Пакет `numaddpackage` замінює собою пакет за замовчуванням.

Для продовження процесу створення інтерфейсу необхідно створити контейнер Java, в який будуть поміщені інші необхідні елементи графічного інтерфейсу. У цій дії контейнер буде створений за допомогою елемента `JFrame`. Контейнер буде поміщений в новий пакет, який буде відображатися в вузлі *Пакети вихідних файлів*. У вікні *Проекти* клацніть правою кнопкою миші вузол NumberAddition і виберіть *Створити-Форма JFrame*. Введіть ім'я класу NumberAdditionUI. Виберіть пакет `numaddpackage`. Натисніть кнопку *Завершити*. Середа IDE створює форму NumberAdditionUI і клас NumberAdditionUI в додатку NumberAddition і відкриває форму NumberAdditionUI в *Конструкторі GUI Builder* (Мал.13).

GUI Builder містить такі елементи: *Область проектування* - основне вікно GUI Builder для створення і редагування форм графічного інтерфейсу Java. Кнопки перемикання між уявленнями *Вихідний код* і *Конструктор* дозволяють переглядати вихідний код класу або графічне представлення компонентів графічного інтерфейсу. Додаткові кнопки панелі інструментів надають швидкий доступ до часто використовуваних команд, наприклад, перемикання між режимами вибору і підключення, вирівнювання компонентів, установка автоматичної зміни розміру для компонентів і попередній перегляд форм.

У вікні *Інспектор* всі компоненти програми (як видимі, так і невидимі) представлені у вигляді дерева ієрархії. Вікно Інспектор також відображає редагований в поточний момент в GUI Builder компонент дерева, а також дозволяє змінювати розташування компонентів на доступних панелях.



Мал.13. Створення фрейму JFrame в GUI Builder IDE NetBeans

Вікно *Палітра* представляє собою список різних компонентів з вкладками для компонентів JFC/Swing, AWT, і JavaBeans, а також диспетчерами компонування. Крім того, існує можливість створення, видалення і зміни порядку проходження категорій, що відображаються у вікні *Палітра*.

Вікно *Властивості* відображає властивості обраного зараз компонента GUI Builder, вікон *Інспектор*, *Проекти* або *Файли*. Додайте в вікні властивостей форми для параметра title заголовок Простий калькулятор, який буде виводитися в заголовку вікна програми (Мал.13).

При натисканні кнопки *Вихідний код* IDE виводить в редакторі вихідний код програми на Java, при цьому автоматично створені засобом GUI Builder розділи коду (захищені блоки) виділяються фоном з сірим кольором. Код в захищених блоках неможливо змінити в поданні *Вихідний код*. Функція редагування доступна в цьому поданні лише для коду на білому фоні вікна редактора. При необхідності зміни коду в захищеному блоці натисніть кнопку *Конструктор* для повернення у вікно GUI Builder, що надає можливість зміни форми. При збереженні змін IDE оновлює вихідний код файлу.

При натисканні на кнопку *Вихідний код* можна побачити автоматично згенерований код, що виводить форму на екран.

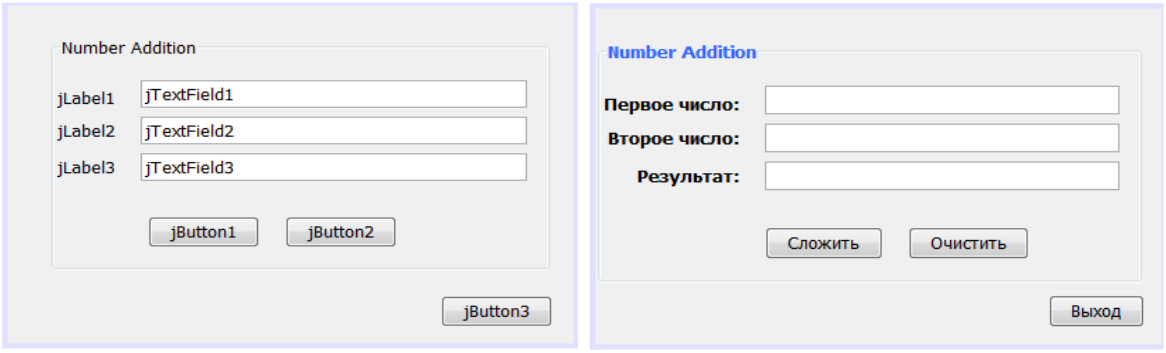
```
package numaddpackage;

// Класс создаваемой формы наследует javax.swing.JFrame
public class NumberAdditionUI extends javax.swing.JFrame {

    /*Создаётся новая форма NumberAdditionUI путём вызова конструктора,
      выполняющего метод initComponents()*/
    public NumberAdditionUI() {
        initComponents();
    }

    /** Метод initComponents() вызывается из конструктора формы
     * и выполняет её инициализацию
     * ПРЕДУПРЕЖДЕНИЕ: НЕ СЛЕДУЕТ модифицировать следующий код.
     * Он всегда генерируется Редактором форм.
     */
    @SuppressWarnings("unchecked")
    private void initComponents() {
        /*Завершение программы при закрытии пользователем окна*/
        setDefaultCloseOperation(javax.swing.WindowConstants
                                 .EXIT_ON_CLOSE);
        /*Установка диспетчера компоновки GroupLayout*/
        javax.swing.GroupLayout layout =
                           new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.
                      Alignment.LEADING).addGap(0, 400, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.
                      Alignment.LEADING).addGap(0, 300, Short.MAX_VALUE)
        );
        pack();
    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new NumberAdditionUI().setVisible(true);
            }
        });
    }

    // Объявления переменных - не изменять
    // окончание объявления переменных
}
```

У порівнянні з програмами, розглянутими в теоретичній частині, в створеному GUI Builder коді є відмінності. Так, конструктор класу `NumberAdditionUI` виконує метод `initComponents()`, в якому відбувається ініціалізація компонентів, розташованих в цьому ж класі. Оскільки метод є закритим, його оператори застосовуються для поточної форми і її компонентів. Відзначимо, що в цьому методі також відбувається налаштування на завершення програми при натисканні кнопки закриття вікна і налаштовується диспетчер компонування

GroupLayout, що дозволяє гнучко налаштовувати розміри і розташування компонентів на формі. Форма створюється в потоці обробки подій, проте для цього використовуються не SwingUtilities, а java.awt.EventQueue. Оскільки поки на формі відсутні компоненти, методи обробки подій не повинні додаватися в форму.

Далі за допомогою вікна *Палітра* зовнішній інтерфейс програми заповнюється панеллю `JPanel` шляхом перетягування елемента *Панель* з вкладки *Палітри Контейнери Swing* (якщо *Палітра* в верхньому правому куті IDE відсутня, виберіть команду з основного меню *Вікна-Палітра*). Виділіть панель `JPanel` клацанням на ній. Перейдіть до вікна *Властивості* і натисніть кнопку з трьома крапками (...) поряд з полем border (межа) для вибору стилю межі. У діалоговому вікні *Border* виберіть *Рамка з написом* зі списку і введіть Number Addition в поле *Заголовок*. Тут же Ви можете змінити колір напису і зробити напівжирним шрифт. Для збереження змін і закриття діалогового вікна натисніть кнопку OK. Після цього додайте три елементи `JLabel` (Напис), три елементи `JTextField` (Текстове поле) і три елементи `JButton` (Кнопка). Третя кнопка `jButton3` повинна знаходиться за межами панелі `jPanel1` (Мал. 14а). Перейдіть до вихідного коду форми. Зверніть увагу на автоматичне додавання операторів створення компонентів, завдання їх назв і включення в конфігурацію диспетчера компонування в методі `initComponents()`. Також компоненти з'явилися в області опису змінних класу NumberAdditionUI.

Перейдіть у вікно *Конструктора*, виконайте подвійне клацання на напису `jLabel1` і змініть властивість тексту на `Перше число:`. Аналогічно змініть текст `jLabel2` на `Друге число:` і jLabel3 - на `Результат:`. Виділіть всі три написи, виконуючи клацання мишею по кожній з них, утримуючи клавішу *Shift*. Зробіть напівжирним шрифт написів, змінивши у вікні *Властивості* параметр font. Залишивши виділеними все три написи, вирівняйте їх по правому краю, натиснувши на кнопку  (Вирівнювання направо в стовпці). Видаліть стандартний текст з текстових полів. Для цього спочатку виконайте клацання на текстовому полі і через деякий час виконайте клацання ще раз. Текст буде виділено і Ви зможете його видалити. Аналогічним способом змініть текст на кнопках `jButton1` на `Скласти`, `jButton2` - на `Очистити` і `jButton3` - на `Вихід`. Альтернативним способом зміни тексту є вибір команди `Змінити текст` з контекстного меню після клацання на кнопці. Тепер готовий графічний інтерфейс повинен виглядати так, як показано на Мал. 14б. Відкрийте вихідний код і вивчіть зміни в методі `initComponents()`.



а) б)

Мал. 14. Створення фрейму з палітрою, написами, текстовими полями і кнопками в GUI Builder IDE NetBeans

Для того щоб кнопки стали функціональними, кожній з них необхідно привласнити обробник подій, який буде реагувати на дії користувача. У нашому випадку потрібно ідентифікувати подію натискання кнопки - шляхом клацання мишею або за допомогою клавіатури. Тому буде використовуватися інтерфейс ActionListener, призначений для обробки подій ActionEvent.

Виконайте клацання правою кнопкою миші по кнопці `Вихід`. В меню оберіть *Події-Action-actionPerformed*. Зверніть увагу на те, що меню містить безліч інших подій, на які може реагувати програма! При виборі події `actionPerformed` IDE автоматично додає інтерфейс `ActionListener` до кнопки `Вихід` і створює метод-обробник, який буде відповідати за обробку методу `actionPerformed(ActionEvent evt)`. В IDE автоматично відкривається вікно *Вихідний код*, де відображається місце вставки дії, яка повинна виконуватися кнопкою при її натисканні (за допомогою миші або клавіатури). Вікно *Вихідний код* має містити наступні рядки:

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent
                                                                    evt) {
        // TODO add your handling code here:
}
```

Тепер додамо код дії, яке має виконувати кнопка `Вихід`. У наведеному вище коді слід замінити рядок `// TODO add your handling code here:` текстом `System.exit(0);`. Готовий код кнопки *Вихід* повинен виглядати наступним чином:

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent
                                                                    evt) {
    System.exit(0);
}
```

Перейдіть у вікно *Конструктора* і виконайте клацання правою кнопкою миші на кнопці `Очистити` (`jButton2`). В меню оберіть *Події-Action-actionPerformed*. Натискання кнопки `Очистити` повинно призводити до видалення всього тексту з будь-яких текстових полів. Готовий вихідний код обробника повинен виглядати наступним чином:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt){
    jTextField1.setText("");
    jTextField2.setText("");
    jTextField3.setText("");
}
```

Обробник події від кнопки `Скласти` (`jButton2`) повинен виконувати три дії. Спочатку прийняти дані, введені користувачем в полях `jTextField1` і `jTextField2`, і перетворити їх з типу `String` в тип `Float`. Потім має проводитися складання двох чисел. Отримана сума повинна бути перетворена в тип `String` і розміщена в поле `jTextField3`. Перейдіть у вікно *Конструктора* і виконайте клацання правою кнопкою миші на кнопці `Скласти` (`jButton1`). В меню оберіть *Події-Action-actionPerform*ed. Додайте наступний код в обробник:

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt){
    // Создание переменных типа float для хранения слагаемых и суммы
    float num1, num2, result;
    // Преобразование типов значений из String в float и присваивание
    // значений переменным слагаемых
    num1 = Float.parseFloat(jTextField1.getText());
    num2 = Float.parseFloat(jTextField2.getText());
    // Сложение и присваивание результата переменной суммы
    result = num1+num2;
    // Вывод суммы в текстовом окне Результат с одновременным
    // преобразованием типа из float в String
    jTextField3.setText(String.valueOf(result));
}
```

Кожен раз при виборі події в меню *Події* IDE автоматично створює *інтерфейс прослуховування подій* (*event listener*) і прив'язує його до компонента. У методі `initComponents()` знайдіть наступний фрагмент:

```
jButton3.setText("Выход");
jButton3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton3ActionPerformed(evt);
    }
});
```

У цьому місці до елементу графічного інтерфейсу, в даному випадку до `jButton3`, додається об'єкт прослуховування подій `ActionListener`. Інтерфейс `ActionListener` має метод `actionPerformed` об'єкта `ActionEvent`, який реалізується шляхом виклику обробника подій `jButton3ActionPerformed`. Тепер ця кнопка реагує на події, що ініціюються користувачем. Кожен раз при натисканні кнопки створюється об’єкт `ActionEvent`, який передається в метод `actionPerformed` інтерфейсу прослуховування подій, і ініціює виконання коду, передбаченого розробником для цієї події в обробнику подій. Як правило, для отримання можливості реагування кожен інтерактивний елемент графічного інтерфейсу повинен бути зареєстрований у якомусь інтерфейсі прослуховування подій і мати пов'язаний обробник подій. Однак IDE забезпечує автоматичну прив'язку інтерфейсу прослуховування, що дозволяє розробнику приділити основну увагу реалізації бізнес-логіки, яка повинна ініціюватися подією.

Запустіть проект. При появі вікна з повідомленням про те, що для проекту NumberAddition не встановлено головний клас, виберіть у якості головного класу `numaddpackage.NumberAdditionUI` і натисніть кнопку OK. Перевірте роботу програми.

Для запуску програми поза IDE виберіть команду *Виконати-Очистити і побудувати головний проект* (Shift-F11). За допомогою провідника файлової системи або Диспетчера файлів перейдіть в каталог NumberAddition/dist. Двічі виконайте клацання на файлі NumberAddition.jar.

3. Завдання

1. Виберіть завдання у наведеній нижче таблиці (Номер варіанта вибирайте за формулою V = (№ mod 16) +1, де № - Ваш порядковий номер в журналі академгрупах.

V	завдання	V	завдання
1	Створити додаток "Гуртожиток", що відображає у вигляді кнопок поверхи гуртожитку, при натисканні на одну з яких відображається план поверху з 8 квартирами-кнопками, при натисканні на одну з яких відображається список мешканців квартири. Передбачити додавання і редагування мешканців	9	Створити додаток "Стипендія", що дозволяє вводити прізвища студентів обраної зі списку спеціальності разом з їх середнім балом за сесію. Додаток має будувати рейтинг для студентів однієї спеціальності і виводити в область тексту відсортовані в порядку убування середнього балу рейтинги, в яких задається з інтерфейсу відсоток кращих студентів виділених жирним шрифтом
2	Створити додаток "Тестування", що відображає питання і кілька відповідей до нього (передбачити одноваріантні і багатоваріантні відповіді). Система повинна пропонувати кілька питань, і в кінці тестування виводити розрахований бал	10	Створити додаток "Прапори країн", що відображає в напису за допомогою об'єкта ImageIcon прапор, випадково виведений з масиву файлів із зображеннями прапорів, і дозволяє вибрати зі списку країну, що має цей прапор. Завдання має повторюватися десять разів з виведенням після закінчення кількості вгаданих варіантів
3	Створити додаток "Список посилань", що дозволяє завантажувати текстовий файл з html-кодом і виводити список всіх посилань, що зустрічаються в цьому коді. Посилання повинні бути без повторів, проте із зазначенням кількості повторень для перевірки	11	Створити додаток "Щоденник" для додавання, редагування та видалення заміток. Замітка відноситься до однієї з категорій, які зберігаються в списку. Для замітки вказуються тема і текст замітки.
4	Створити додаток "Геометричні фігури", який обчислює периметр і площу для квадрата, трикутника, прямокутника і трапеції по необхідним даним, які вводить користувач	12	Створити додаток "Конвертор валют". Користувач вводить суму, вибирає з першого списку вихідну валюту, потім з другого - результуючу і отримує еквівалент в іншій валюті

V	завдання	V	завдання
5	Створити додаток "Тригонометричні функції", що дозволяє обчислювати тригонометричні функції sin, cos, tg заданих кутів від 0 до 90º з кроком 5º. Інтерфейс повинен відображати в першому стовпці значення градусів або відповідні значення радіанів (вибираються перемикачем), а в другому стовпці - значення функцій (вибираються зі списку)	13	Створити додаток "Геометричні тіла", що обчислюють площу поверхні і об'єм для куба, прямокутного паралелепіпеда, кулі і прямого кругового конуса по необхідним даним, введеним користувачем
6	Створити додаток "Телефонний довідник", що дозволяє додавати, редагувати і видаляти записи: прізвище, ім'я, по-батькові - номер телефону. Передбачити можливість пошуку записів по прізвищу і по номеру телефону	14	Створити додаток "Покупки", що дозволяє вводити назву, ціну та кількість куплених товарів. Додаток має забезпечувати збереження введених даних в файл і зчитування їх з файлу, а також розрахунок суми покупки
7	Створити додаток "Пошук чисел", що дозволяє знаходити в масиві 15x15 числа, які є ступенем або 2, або 3. Створіть інтерфейс програми: в таблиці 15x15 числа отримати випадковим чином; створити кнопки виконуваних дій; результат дій підсвічувати кольором; в написі виводити кількість знайдених чисел	15	Створити додаток "Перетворення величин", що дозволяє перетворювати введене значення градусів Цельсія в градуси Фаренгейта, кілометрів на милі, кілограмів у фунти, барів в Паскалі, літрів в галони
8	Створити додаток "Бінарний калькулятор", що дозволяє виконувати операції NOT, AND, OR і XOR над двома бітовими числами розмірів 8 або 16 або 32 біта	16	Створити додаток "Арифметичний калькулятор", що дозволяє виконувати операції додавання, віднімання, множення і ділення. Передбачити можливість обчислення виразу, що складається з декількох операторів, при цьому введення знака будь-якої операції або знака "дорівнює" повинен супроводжуватися обчисленням і відображенням проміжного результату. Передбачити можливість редагування і скасування останньої операції.

2. Розробіть програму, що реалізовує обране за варіантом завдання, на мові Java з графічним інтерфейсом користувача з використанням бібліотек класів AWT і Swing з вручну виначеними компонентами.
3. Для розміщення компонентів використовуйте найбільш відповідний диспетчер компонування або їх комбінацію.
4. Передбачте в програмі відповідне меню команд і по можливості, використання текстових полів, радіокнопок, прапорців і списків.
5. Передбачте захист від невірних дій користувача з виведенням повідомлень за допомогою компонента JOptionPane.

6. Повторіть розробку програми, використовуючи GUI Builder IDE NetBeans.
7. Наведіть код обох проектів і скріншоти з інтерфейсом розробленої програми в звіт.

4. Контрольні питання

1. Що називають додатком з графічним інтерфейсом користувача? Дайте визначення компонентам (віджетам). Назвіть бібліотеки Java, які надають засоби створення додатків з графічним інтерфейсом користувача.
2. Що являють собою технології AWT і Swing? Опишіть їх призначення і відмінності. У яких пакетах містяться класи і інтерфейси цих бібліотек?
3. Опишіть ієрархію класів і інтерфейсів бібліотек Swing і AWT. Поясніть її, вказавши великовагові контейнери, легковагові контейнери і компоненти, а також поясніть можливість спільного використання засобів цих бібліотек.
4. Що являє собою компонент і контейнер? Назвіть контейнери Swing верхнього рівня і наведіть приклади легковагових контейнерів Swing. У чому їх відмінність? На базі якого шаблону проектування побудовані контейнери Swing?
5. Опишіть призначення панелей контейнерів верхнього рівня і легковагового контейнера JInternalFrame.
6. Опишіть зазвичай використовувані оператори в конструкторі створення форми. Які існують варіанти конфігурації завершення роботи програми?
7. Які засоби пропонує Swing для запуску об'єкту, що реалізує графічний інтерфейс, в окремому потоці виконання? Які переваги забезпечує такий підхід?
8. Опишіть механізм обробки подій, що використовується в Swing та AWT. Що є джерелом події, слухачем події, подією? Де розташований метод обробки подій? Наведіть приклади подій.
9. Назвіть події, які генерують: кнопка - при натисканні на неї, текстове поле - при переміщенні курсору між символами, список - при виборі елемента, прапорець - при установці, елемент меню - при виборі за допомогою клавіатурної комбінації і при виборі мишею.
10. На основі якого шаблону проектування реалізована модель обробки подій в Swing та AWT? Поясніть елементи цього шаблону. Яким методом здійснюється реєстрація слухача події в джерелі?
11. Перерахуйте конструктори JButton. Наведіть приклад створення кнопки з текстом і з графічним зображенням.
12. Опишіть оператори організації обробки подій при натисканні на кнопку. Для чого використовується команда дії? Як визначити команду дії події, що відбулася?
13. Що виконують диспетчери компонування AWT-Swing? Опишіть відомі Вам диспетчери компонування і наведіть приклади завдань, коли вони можуть бути використані.

14. Опишіть диспетчер компонування BorderLayout. Наведіть приклад коду, який розміщує компоненти в різних областях форми. Який контейнер має цей диспетчер за замовчуванням?
15. Опишіть диспетчер компонування GridLayout. Наведіть приклад коду, що використовує цей диспетчер. Як підтримується динамічна зміна кількості рядків або стовпців?
16. Наведіть приклад використання комбінації диспетчерів компонування для створення стандартної форми з фіксованою рядком меню і кнопками, притиснутими до правого краю. Чим GridLayout з одним рядком відрізняється від BoxLayout?
17. Опишіть оператори організації обробки подій при натисканні на клавішу Enter після введення в текстове поле. Як отримати введений в поле текст? Наведіть організацію роботи текстового поля з Буфером обміну.
18. Опишіть відомі Вам конструктори JTextField. Наведіть оператори організації обробки подій при зміні положення курсору в текстовому полі. Назвіть спадкоємців абстрактного класу javax.swing.text.JTextComponent.
19. Опишіть оператори організації обробки подій при виборі елемента JCheckBox і елемента JRadioButton. Наведіть приклад організації радіокнопок в групу. Які методи дозволяють динамічно перемальовувати панель?
20. Опишіть оператори організації обробки подій при виборі елемента JList. Які режими вибору підтримує цей компонент? Як відобразити довгий список на обмеженій площі?
21. Опишіть оператори організації обробки подій при виборі елемента JComboBox. Як відобразити довгий список на обмеженій площі?
22. Опишіть оператори організації меню. Як встановити клавіатурні комбінації вибору меню? Як згрупувати елементи меню?
23. Опишіть організацію вибору файлу з використанням компонента JFileChooser. Який клас допомагає реалізувати фільтрацію відображуваних файлів по розширенню?
24. Наведіть приклад організації виклику модального діалогового вікна для повідомлення користувачу про помилку, що виникла при виконанні програми.
25. Опишіть статичні методи класу JOptionPane, що відображають діалогові вікна різних типів і поясніть призначення їх параметрів.
26. Опишіть інтерфейс NetBeans IDE GUI Builder і способи роботи з ним. Яка частина коду автоматично генерується?
27. У чому полягають особливості створюваного GUI Builder коду? Як він організовує обробку подій?
28. Як скомпілювати проект з можливістю його запуску поза IDE?

5. Література

1. Шілдт Г. SWING: Керівництво для початківців: Пер. з англ. -М .: ТОВ "І.Д. Вільямс", 2007. -704с.

2. Портянкін І.А Swing: Ефектні користувальницькі інтерфейси - Бібліотека програміста. СПб .: Пітер, 2005. -528 с.
3. Гал С., Певек Т., Кастерер Р., Кіген П. Введення в розробку графічного інтерфейсу - http://NetBeans.org/kb/docs/java/gui-functionality_ru.html.