Міністерство освіти і науки України
Сумський державний університет

МЕТОДИЧНІ ВКАЗІВКИ

для виконання лабораторних робіт на тему
«Використання системи Swing в Java»
із дисциплін
«Програмування мобільних комп'ютерних систем»,
«Прикладне програмування в телекомунікаційних системах»,
«Програмування мобільних пристроїв телекомунікацій»
для студентів спеціальностей
6.171 "Електроніка",
6.172 "Телекомунікації та радіотехніка"
денної форми навчання

Суми
Видавництво СумДУ
2021

Методичні вказівки для виконання лабораторних робіт на тему «Використання системи Swing в Java» із дисциплін «Програмування мобільних комп'ютерних систем», «Прикладне програмування в телекомунікаційних системах», «Програмування мобільних пристроїв телекомунікацій» / Укладач О.Є. Горячев – Суми: Вид-во СумДУ, 2021. – 67 с.

Кафедра електроніки і комп'ютерної техніки

1 Мета роботи

Метою роботи є набуття навичок програмування графічних додатків Java з використанням елементів управління AWT.

2 Склад робочого місця.

IBM-сумісний персональний комп'ютер (ПК).

Програмне забезпечення: операційна система Windows, Java 2 SDK версії 6.0 і вище, пакет NetBeans IDE 6.1 і вище.

3 Короткі теоретичні відомості

3.1. Основні компоненти Swing

3.1.1. Створення та ініціалізація об'єкта класу String

Компоненти Swing можна розділити на наступні типи:

- Контейнери верхнього рівня (класи **JWindow**, **JFrame**, **JDialog** і **JApplet**);
- Спеціалізовані контейнери (класи **JInternalFrame**, **JLayeredPane**, **JRootPane** і **JOptionPane**);
- Контейнери загального призначення (класи **JPanel**, **JScrollPane**, **JSplitPane**, **JTabbedPane** і **JToolBar**);
- Компоненти управління (класи **JButton**, **JCheckBox**, **JRadioButton**, **JToggleButton**, **JComboBox**, **JList**, **JMenuBar**, **JMenu**, **JMenuItem**, **JCheckboxMenuItem**, **JRadioButtonMenuItem**, **JSeparator** і **JSlider**);
- Нередаговані інформаційні компоненти (класи **JLabel**, **JProgressBar** і **JToolTip**);
- Редаговані інформаційні компоненти (класи **JColorChooser**, **JFileChooser**, **JTable**, **JTree**, **JTextField**, **JPasswordField**, **JTextArea**, **JEditorPane** і **JTextPane**).

На відміну від компонентів AWT, компоненти системи Swing здатні працювати тільки за моделлю делегування подій.

3.1.2. Контейнери верхнього рівня і спеціалізовані контейнери

Так само, як і для AWT, для створення вікон графічних додатків використовується не клас **JWindow**, а клас **JFrame** (вікна, створювані класом **JWindow** не містять найменування вікна і кнопок управління вікном). Додатки з графічним інтерфейсом використовує, щонайменш, один фрейм.

Для створення вікон, які залежать від іншого вікна (наприклад, зникають, коли згортається вікно, в якому вони використовуються) застосовуються діалогові вікна класу **JDialog**.

Будь-яка програма, яка використовує компоненти Swing, містить, принаймні, один контейнер верхнього рівня. Цей контейнер є коренем ієрархії контейнерів, що містять всі компоненти Swing.

Як правило, окремий графічний додаток має, принаймні, одну ієрархію контейнерів, в якій коренем є **JFrame**. Діалогове вікно або аплет також утворюють ієрархію контейнерів, коренем якої є JDialog або JApplet. Наприклад, якщо додаток містить одне головне вікно і два діалогових вікна, то він містить три ієрархії контейнерів.

3.1.2.1. Коренева панель

Кожен контейнер верхнього рівня базується на проміжному, прихованому, контейнері, званому кореневої панеллю (root pane). Коренева панель визначена в класі **JRootPane**.

Сама коренева панель зазвичай не використовується, а використовуються її компоненти, які коренева панель (клас) надає кадру (або іншому контейнеру верхнього рівня). Коренева панель містить такі компоненти:

- шарувата панель (***layered pane***);
- панель вмісту (***content pane***);
- рядок меню (***menu bar***) - необов'язковий компонент;
- скляна панель (***glass bar***).

Єдиним обов'язковим контейнером верхнього рівня є панель вмісту.

3.1.2.2. Панель вмісту

Панель вмісту містить всі компоненти Swing (кнопки, написи, текстові поля і т.д.). Оскільки для контейнерів верхнього рівня вміст вікна визначається за допомогою **JRootPanel** і має, на відміну від вікон AWT, визначатися вручну, для додавання компонент або установки менеджера компоновки використовуються не методи `add();` і `setLayout();`, а методи отримання і установки панелі вмісту:

```
public Container getContentPane ();
```

і

```
public void setContentPane (Container contentPane);
```

які визначені в класах **JFrame**, **JDialog**, **JApplet** і **JInternalFrame**.

Так, додавання текстового поля у фрейм, діалогове вікно, аплет або внутрішній фрейм з установкою менеджера компоновки FlowLayout виглядає наступним чином:

```
Container contentPane = getContentPane ();
JTextField inputField = new JTextField (15);
contentPane.setLayout (new FlowLayout ());
contentPane.add (inputField);
```

Слід зазначити, що для всіх контейнерів верхнього рівня (включаючи **JApplet**) менеджером компонування за замовчуванням є **BorderLayout**.

До контейнера верхнього рівня може бути додано рядок меню (***menu bar***). Рядок меню також розташовується всередині головного контейнера, але за межами панелі вмісту.

3.1.2.3. клас JFrame

Фрейм в Swing створюється за допомогою конструкторів

```
JFrame ();
```

або

```
JFrame (String name);
```

де **name** - ім'я фрейму.

Для фрейма можна використовувати наведені вище методи отримання і установки кореневої панелі, шаруватої панелі, панелі вмісту, рядків меню і скляної панелі, причому має бути отримано, принаймні, значення панелі вмісту.

Іншою відмінністю **JFrame** від **Frame** є наявність у **JFrame** властивості, що визначає операцію закриття вікна за замовчуванням. Для **Frame** за замовчуванням нічого не відбувається, якщо робиться спроба закрити вікно, а **JFrame** ховається. Метод

```
public void setDefaultCloseOperation (int operation)
```

класу **JFrame** дозволяє визначити три операції, які можуть виконуватися при закритті вікна **JFrame**:

• **DO_NOTHING_ON_CLOSE** - поводиться так само, як AWT **Frame**;

• **HIDE_ON_CLOSE** - поведінка за умовчанням (для того, щоб вікно знову з'явилося на екрані, необхідно використовувати метод **setVisible (true);**

• **DISPOSE_ON_CLOSE** - вікно закривається.

3.1.2.4. Діалогові вікна

Діалогові вікна можна створювати або за допомогою класу **JDialog**, або використовуючи клас **JOptionPane**.

Основними конструкторами класу **JDialog** є:

```
JDialog ();
JDialog (Dialog owner);
JDialog (Dialog owner, boolean modal);
JDialog (Dialog owner, String title);
JDialog (Dialog owner, String title, boolean modal);
JDialog (Frame owner);
JDialog (Frame owner, boolean modal);
JDialog (Frame owner, String title);
JDialog (Frame owner, String title, boolean modal);
```

Перший конструктор створює немодального діалогове вікно без імені. Параметри **owner**, **title** і **modal** в інших конструкторах задають відповідно батька даного діалогового вікна, ім'я вікна і тип вікна (**true** - модальне вікно або **false** - немодального вікно).

Вікно класу **JDialog** має ті ж риси, що **JFrame** і пряме використання об'єктів класу **JDialog** аналогічно прямому використанню об'єкта **JFrame**.

Клас **JOptionPane** полегшує вивід стандартних діалогових або інформаційних вікон.

Для класу **JOptionPane** визначені наступні основні конструктори:

```
JOptionPane ();
```

```
JOptionPane (Object message, int messageType,
int optionType, Icon icon, Object [] options,
Object initialValue);
```

Крім цього визначені конструктори, які містять відповідно тільки перший параметр, тільки перші два параметра, тільки перші три параметри, тільки перші чотири параметри, тільки перші п'ять параметрів другого конструктора.

Все використання цього класу зазвичай зводяться до виклику одного з його методів:

• **`static int showConfirmDialog (Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon)`** - задає питання на який потрібно відповідь типу **Yes**, **No** або **Cancel**. Існують також наступні три методи **showConfirmDialog ()**: метод, що містить перші два параметра, метод, що містить перші чотири параметри і метод, що містить перші п'ять параметрів, наведеного вище методу;

• **`static Object showInputDialog (Component parentComponent, Object message, String title, int messageType, Icon icon, Object [] selectionValues, Object initialValue)`** - дозволяє ввести деякий текст. Існують також наступні п'ять методів **showInputDialog ()**: метод, що містить другий параметр, метод, що містить другий і сьомий параметри, метод, що містить перші два параметра, метод, що містить перший, другий і сьомий параметри, і метод, що містить перші три параметри і сьомий параметр ;

• **`static void showMessageDialog (Component parentComponent, Object message, String title, int messageType, Icon icon)`** - дозволяє вивести деякий повідомлення. Існують також наступні два методи **showMessageDialog ()**: метод, що містить перші два параметра і метод, що містить перші чотири параметри;

• **`static int showOptionDialog (Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object [] options, Object initialValue)`** - об'єднує функції перерахованих вище діалогових вікон.

Параметри в наведених конструкторах і методах мають наступний сенс:

• **parentComponent** - визначає компонент, який є батьком даного діалогового вікна (якщо цей параметр дорівнює ***null***, то в якості батька передбачається фрейм за умовчанням);

• **message** - описову повідомлення, що поміщається в діалогове вікно (слід мати на увазі, що це об'єкт класу **Object**);

• **selectionValues** - масив об'єктів, що розглядається як набір повідомлень;

• **icon** - зображення, що виводиться в діалозі (значення за замовчуванням визначається значенням параметра **messageType**);

• **messageType** - визначає тип повідомлення. Можливі наступні значення: **`ERROR_MESSAGE`**, **`INFORMATION_MESSAGE`**, **`WARNING_MESSAGE`**, **`QUESTION_MESSAGE`** і **`PLAIN_MESSAGE`**;

• **optionType** - визначає набір кнопок, які будуть виведені в нижній частині діалогового вікна: **`DEFAULT_OPTION`**, **`YES_NO_OPTION`**, **`YES_NO_CANCEL_OPTION`** і **`OK_CANCEL_OPTION`**

• **options** - більш детальний опис набору кнопок, які будуть виведені в нижній частині діалогового вікна (якщо заданий об'єкт класу **Component**, він прямо додається в нижню частину діалогового вікна, якщо об'єкт типу **Icon**, створюється кнопка класу **JButton**, для об'єктів інших класів проводиться перетворення об'єкта в рядок, який виводиться на кнопці класу **JButton**);

• **title** - ім'я діалогового вікна;

• **initialValue** - вводиться значення за замовчуванням.

Для методів і припустимі наступні можливі значення, що повертається: **`YES_OPTION`**, **`NO_OPTION`**, **`CANCEL_OPTION`**, **`OK_OPTION`** і **`CLOSED_OPTION`**.

3.1.3. Загальноцільові контейнери

3.1.3.1. клас JPanel

Клас **JPanel** є варіантом класу **Panel** в AWT. Чотири конструктора класу:

```
public JPanel ();
public JPanel (LayoutManager layout);
public JPanel (boolean isDoubleBuffered);
public JPanel (LayoutManager layout,
boolean isDoubleBuffered);
```

забезпечують створення об'єкта **JPanel** відповідно під управлінням менеджера компоновки **FlowLayout**, під керуванням заданого менеджера компоновки, з вбудованою підтримкою подвійної буферизації і, нарешті, під керуванням заданого менеджера компоновки і з вбудованою підтримкою подвійної буферизації.

Робота з об'єктами класу **JPanel** не відрізняється від роботи з об'єктами класу **Panel**, за винятком того, що панелі додаються не у фрейм, а в панель вмісту фрейма.

3.1.3.2. Клас JTabbedPane

Панель з вкладками (***tabbed pane***) введена в Swing замість використовувався в AWT менеджера **CardLayout**, тобто менеджер компонування **CardLayout** в Swing не реалізований. Кожна вкладка має заголовок. Коли користувач вибирає вкладку, її вміст стає видимим. Тільки одна з вкладок може бути обрана одночасно. Панель з вкладкамі зазвичай використовується для установки параметрів конфігурації.

Панель з вкладками реалізована класом **JTabbedPane**, який має такі конструктори

```
JTabbedPane ();
JTabbedPane (int tabPlacement);
JTabbedPane (int tabPlacement, int tabLayoutPolicy);
```

Перший конструктор створює нову панель з вкладками, в якій заголовки вкладок розташовані зверху, а другий дозволяє визначити розташування заголовків вкладок:

JTabbedPane.TOP - зверху,

JTabbedPane.BOTTOM - знизу,

JTabbedPane.RIGHT - вправо або

JTabbedPane.LEFT - вліво.

Третій конструктор додатково задає правило розташування вкладок:

JTabbedPane.WRAP_TAB_LAYOUT - вкладки розташовуються в кілька рядків, якщо вони не поміщаються в одному рядку - значення за замовчуванням і

JTabbedPane.SCROLL_TAB_LAYOUT - вкладки прокручуються, якщо вони не поміщаються в одному рядку.

Формування в програмі панелі з вкладками виконується за допомогою такої процедури:

1. Створити об'єкт **JTabbedPane**.

2. Для додавання кожної вкладки в панель викликати один з методів **addTab ():**

```
public void addTab (String title, Icon icon,
Component component, String tip);
public void addTab (String title, Icon icon,
Component component);
public void addTab (String title, Component component);
```

Аргументи цих методів визначають заголовок вкладки **title**, компоненту **component**, яку вона має, зображення **icon** і підказку **tip**.

3. Додати панель з вкладками в панель змісту вікна.

Для відстеження вибору тієї чи іншої вкладки використовується блок прослуховування **ChangeListener**, який додається або видаляється з допомогою методів

```
public void addChangeListener (ChangeListener l);
```

і

```
public void removeChangeListener (ChangeListener l);
```

класу **JTabbedPane**. В інтерфейсі **ChangeListener** визначено єдиний метод

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

в якому за допомогою методів

```
public Component getSelectedComponent ();
```

або

```
public int getSelectedIndex ();
```

класу **JTabbedPane** можна отримати обраний компонент або індекс обраного компонента.

3.1.3.3. Клас JSplitPane

Розщеплені панелі (клас **JSplitPane**) використовуються для поділу виведеної панелі на дві компоненти. Конструктори класу

```
public JSplitPane ();
public JSplitPane (int newOrientation);
public JSplitPane (int newOrientation,
boolean newContinuousLayout);
public JSplitPane (int newOrientation,;
Component newLeftComponent, Component
newRightComponent);
public JSplitPane (int newOrientation,
boolean newContinuousLayout, Component newLeftComponent,
Component newRightComponent);
```

задають в параметрі **newOrientation** вирівнювання зліва направо з використанням константи **`JSplitPane.HORIZONTAL_SPLIT`** або зверху вниз з використанням константи **`JSplitPane.VERTICAL_SPLIT`**, в параметрі **newContinuousLayout** безперервну перерисовку компонент при переміщенні роздільника (значення ***true***) або перемальовування компонент після закінчення переміщення роздільника (значення ***false***), а в параметрах **newLeftComponent** і **newRightComponent** - компоненти розщепленої панелі.

Положення роздільника в панелі задається за допомогою методу

```
public void setDividerLocation (int location),
```

який задає в пікселях позицію ***x*** або ***y*** роздільника (в залежності від вертикальної або горизонтальної орієнтації панелей). Метод

```
public void setOneTouchExpandable (boolean newValue)
```

дозволяє задати на розділовій смузі кнопки (зі стрілками) для швидкого згортання і розгортання однієї з панелей розщепленої панелі.

3.1.4. Значки, написи і кнопки

3.1.4.1. Інтерфейс Icon і клас ImageIcon

Значки (**icons**) реалізуються в Swing за допомогою інтерфейсу **Icon** (насправді вони не є компонентами і їх можна використовувати практично з усіма компонентами Swing). Реалізацією інтерфейсу **Icon**, що створює значок із зображення, є клас **ImageIcon**. Двома основними конструкторами цього класу є:

```
ImageIcon (String filename);
ImageIcon (URL url);
```

Перша форма використовує зображення під ім'ям **filename** (використовується в графічних додатках) а друга форма - в ресурсі, розташованому по URL-адресою **url** (використовується в апплетах Swing)

На відміну від класу **Image**, в класі **ImageIcon** зображення завантажується попередньо з використанням методів класу **MediaTracker**.

Для створення власних зображень можна безпосередньо реалізовувати інтерфейс **Icon**.

3.1.4.2. клас JLabel

Написи Swing є об'єктами класу **JLabel**. Цей об'єкт може відображати тексти і / або значки. Основними його конструкторамі є:

```
JLabel (Icon image);
JLabel (String text);
JLabel (String text, Icon image, int horizontalAlignment);
```

У цих конструкторах **text** і **image** - текст і значок, який використовується в написи. Параметр **horizontalAlignment** определяет вирівнювання і має значення **LEFT**, **RIGHT** або **CENTER**. Ці константи визначені в інтерфейсі **SwingConstants**, поряд з кількома іншими, використовуваними класами системи Swing.

Значок і текст, пов'язаний з написом, можна зчитувати і записувати наступними методами:

```
public Icon getIcon ();
public String getText ();
public void setIcon (Icon image);
public void setText (String text);
```

Для обробки подій в об'єктах класу **JLabel** можна використовувати інтерфейс **MouseListener** і клас **MouseEvent**.

3.1.4.3. клас AbstractButton

Кнопки Swing є підкласами класу **AbstractButton**. Цей клас містить багато методів, які дозволяють управляти поведінкою кнопок, прапорців та перемикачів. Наприклад, можна визначати різні значки для відображення компонента, коли він віджатий (***disabled***), натиснутий (***pressed***), або обраний (***selected***). Деякий значок можна використовувати як значок "наїзду" (***rollover***), котрий відображається, коли курсор миші встановлено поверх цього компонента. Нижче наведені описи методів, які керують такою поведінкою:

```
public void setDisabledIcon (Icon image);
public void setPressedIcon (Icon image);
public void setSelectedIcon (Icon image);
public void setRolloverIcon (Icon image);
```

Текст, пов'язаний з кнопкою, можна читати і записувати за допомогою наступних методів:

```
public String getText ();
public void setText (String text);
```

У класі **AbstractButton** визначені також методи

```
public int getMnemonic ();
```

і

```
public void setMnemonic (int mnemonic);
```

відповідно для отримання і установки «гарячих» клавіш. Як параметр при установці «гарячої» клавіші задається значення з статичних властивостей класу **KeyEvent**. Ці властивості мають загальне ім'я **VK_XXX**, де **XXX** - ім'я однієї з клавіш, наприклад, **VK_D**.

При натисканні кнопки конкретні підкласи **AbstractButton** генерують події **Action**. Блоки прослуховування реєструють і скасовують регістрацію для цих подій за допомогою таких методів:

```
void addActionListener (ActionListener l);
void removeActionListener (ActionListener l);
```

3.1.4.4. клас JButton

Клас **JButton** є підкласом класу **AbstractButton** і забезпечує функціональні можливості кнопки. Цей клас дозволяє зв'язати з кнопкою зображення, текст або і те й інше. Деякі з його конструкторів:

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

Тут **text** і **image** - напис і зображення, які використовуються для кнопки.

Клас **JButton** успадковує властивості інтерфейсу **SwingConstants**, а також властивості та методи класу **AbstractButton**.

Для обробки подій в об'єктах класу **JButton** можна використовувати інтерфейс **ActionListener** і клас **ActionEvent**.

3.1.5. Прапорці та перемикачі

Клас **JCheckBox**, який забезпечує функціональні можливості прапорця, є конкретною реалізацією класу **AbstractButton**. Деякі з його конструкторів:

```
JCheckBox (Icon image);
JCheckBox (Icon image, boolean state);
JCheckBox (String text);
JCheckBox (String text, boolean state);
JCheckBox (String text, Icon image);
JCheckBox (String text, Icon image, boolean state);
```

У цих конструкторах використовуються наступні параметри: **image** - зображення для прапорця, **text** - напис. Якщо state має значення true, то прапорець спочатку вибраний.

Стан прапорця може бути змінено за допомогою методу

```
void public void setSelected (boolean b);
```

а отримати стан прапорця можна за допомогою методу

```
public boolean isSelected ();
```

Перемикачі підтримуються класом **JRadioButton**, який також є конкретною реалізацією класу **AbstractButton**. Нижче наведені деякі з його конструкторів:

```
JRadioButton (Icon image);
JRadioButton (Icon image, boolean state);
JRadioButton (String text);
JRadioButton (String text, boolean state);
JRadioButton (String text, Icon image);
JRadioButton (String text, Icon image, boolean state);
```

де параметри мають таке ж значення, що і для класу **JCheckBox**.

Перемикачі повинні бути об'єднані в групу, де в кожен момент може бути обраний лише один елемент. Наприклад, якщо користувач клацає по перемикачу, який знаходиться в групі, попередньо натиснутий перемикач в цій групі автоматично скидається. Щоб створити групу кнопок, створюється об'єкт класу **ButtonGroup**. Для цієї мети використовується єдиний конструктор класу

```
public ButtonGroup ().
```

Елементи до групи перемикачів додаються за допомогою методу

```
public void add (AbstractButton a),
```

а кількість перемикачів в групі можна отримати за допомогою методу

```
public int getButtonCount ().
```

Батьківським класом і для **JCheckBox** і для **JRadioButton** є клас **JToggleButton**, який не має еквівалента в AWT. Він веде себе як об'єкт класу Button, який після натискання на нього залишається в натиснутому стані.

Для обробки подій в об'єктах класів **JCheckBox** і **JRadioButton** можна використовувати інтерфейс **ItemListener** і клас **ItemEvent**.

3.1.6. Списки

3.1.6.1. Клас JComboBox

Замість класу **Choice** в AWT, в системі Swing забезпечує комбіноване поле (***combo box***) - комбінація текстового поля і списку (клас **JComboBox**). Комбіноване поле зазвичай відображає один вхід (елемент) списку. Однак воно може також відображати і список що розкривається, який дає можливість користувачу вибирати різні елементи. Ви також можете ввести з клавіатури своє значення елемента списку в текстове поле. Два найпоширеніших конструктора:

```
JComboBox ();
JComboBox (Object [] items);
```

дозволяють створити порожній список з вибором або задати масив об'єктів даного списку.

Елементи додаються до списку вибору за допомогою методу

```
void addItem (Object obj);
```

де **obj** - об'єкт, який буде додано до комбінованого полю.

Решта методів класу аналогічні відповідним методам класу **Choice**, однак додані методи

```
public void setEditable (boolean aFlag);
```

і

```
public boolean isEditable ();
```

що дозволяють встановити можливість введення значення в список і перевірити прапорець можливості введення свого значення в список.

Для обробки подій, пов'язаних з комбінованим списком, можна використовувати або блоки прослуховування **ItemListener**, або блоки прослуховування **ActionListener**.

3.1.6.2. клас JList

Аналогом класу **List** в Swing є клас **JList**. Два найпоширеніших конструктора цього класу

```
JList ();
JList (Object [] items);
```

дозволяють задати або порожній список, або список, який містить задані об'єкти (зверніть увагу, що елементами списків в Swing можуть бути не тільки рядки, але довільні об'єкти).

Набір методів в класі **JList** істотно розширено в порівнянні з класом **List** (так, наприклад, є можливість задавати колір тексту і фону для виділених елементів) і, крім того, замість інтерфейсу **ItemListener** для списків **JList** використовується інтерфейс **ListSelectionListener**, блок прослуховування якого додається і видаляється з допомогою методів

```
public void addListSelectionListener
(ListSelectionListener listener);
```

і

```
public void removeListSelectionListener (
ListSelectionListener listener);
```

класу **JList**.

Метод

```
public void valueChanged (ListSelectionEvent e);
```

інтерфейсу обробляє події, пов'язані з вибором елементів списку, а метод

```
public Object [] getSelectedValues ();
```

класу **JList** дозволяє отримати список обраних об'єктів.

3.1.6.3. клас JSpinner

Клас **JSpinner** визначає список що «обертається». Цей клас схожий на клас **JComboBox**, але однорядковий список можна прокручувати в двох напрямках: вгору і вниз за допомогою двох кнопок праворуч від поля вибору.

Для класу JSpinner визначені два конструктора:

```
JSpinner ();
```

і

```
JSpinner (SpinnerModel model);
```

Другий конструктор створює об'єкт для заданої моделі списку що «обертається». У пакеті визначені класи для найбільш уживаних моделей списку що «обертається»: **SpinnerListModel** (для списків), **SpinnerNumberModel** (для чисел) і **SpinnerDateModel** (для дат).

Встановити або отримати модель можна або за допомогою методів

```
void setModel (SpinnerModel model);
```

і

```
SpinnerModel getModel ();
```

Змінити виведене в поле значення можна за допомогою методу

```
void setValue (Object value),
```

а отримати поточне значення поля можна за допомогою методу

```
Object getValue ();
```

класу **JSpinner**.

Встановити конкретну модель для списків можна за допомогою наступних конструкторів:

```
SpinnerListModel ();
SpinnerListModel (List values);
SpinnerListModel (Object [] values);
```

Перший конструктор створює модель для порожнього списку, другий для списку об'єктів колекції **List**, а третій для масиву об'єктів класу **Object**.

Метод

```
void setValue (Object obj)
```

змінює елемент в полі виведення і повідомляє блок прослуховування **ChangeListener**. Методи

```
Object getValue ();
Object getNextValue ();
Object getPreviousValue ();
```

дозволяють отримати відповідно поточне значення поля виведення, наступне значення поля виведення і попереднє значення поля виведення. Якщо поточний елемент - останній, метод **getNextValue ()** повертає null, а якщо поточний елемент - перший, метод **getPreviousValue ()** повертає null.

Основними конструкторами моделі чисел є:

```
SpinnerNumberModel ();
SpinnerNumberModel (int value, int minimum,
int maximum, int stepSize);
```

де **value** - виведене в поле виведення значення, **minimum** - мінімальне значення списку, **maximum** - максимальне значення списку, **stepSize** - крок списку. Порожній конструктор встановлює список з нульовим значенням в полі виведення, без обмежень на максимальне і мінімальне значення і з кроком, рівним 1.

Значення вказаних параметрів також можна встановити за допомогою методів

```
void setValue (Object value);
void setMinimum (Comparable minimum);
void setMaximum (Comparable maximum);
void setStepSize (Number stepSize);
```

а отримати їх значення - за допомогою методів

```
Object getValue ();
Object getNextValue ();
Object getPreviousValue ();
Comparable getMinimum ();
Comparable getMaximum ();
Number getStepSize ();
```

Для роботи з датами визначені наступні конструктори:

```
SpinnerDateModel ();
SpinnerDateModel (Date value, Comparable start,
Comparable end, int calendarField);
```

де **value** - значення, що виводиться дати, **start** і **end** - початкове і кінцеве значення дати, а допустимі значення для **calendarField**:

• **Calendar.ERA** - ера (значення **GregorianCalendar.BC** - до нашої ери або **GregorianCalendar.AD** - наша ера);

• **Calendar.YEAR** і **Calendar.MONTH** - значення року і місяця (від 0 до 11);

• **Calendar.WEEK_OF_YEAR** і **Calendar.WEEK_OF_MONTH** - тиждень року і тиждень місяця;

• **Calendar.DAY_OF_MONTH** і **Calendar.DAY_OF_YEAR** - день місяця і день року;

• **Calendar.DAY_OF_WEEK** і **Calendar.DAY_OF_WEEK_IN_MONTH** - день тижня (від 1 до 7) і день тижня в місяці (1-7 дні місяця - 1 тиждень, 8-14 - 2 тиждень і т.д.);

• **Calendar.AM_PM** - до полудня - значення **Calendar.AM** або після полудня - значення **Calendar.PM**;

• **Calendar.HOUR** і **Calendar.HOUR_OF_DAY**, - годину в форматі від 0 до 12 або від 0 до 24;

• **Calendar.MINUTE**, **Calendar.SECOND** і **Calendar.MILLISECOND** - хвилина, секунда і мілісекунда.

Значення вказаних параметрів також можна встановити за допомогою методів

```
void setValue (Object value);
void setStart (Comparable start);
void setEnd (Comparable end)
void setCalendarField (int calendarField);
```

а отримати їх значення - за допомогою методів

```
Object getValue ();
Object getNextValue ();
Object getPreviousValue ();
Comparable getStart ();
Comparable getEnd ();
Number getCalendarField ();
```

метод

```
Date getDate ();
```

повертає поточний елемент в послідовності дат у вигляді об'єкта класу **Date**.

Для обробки подій в об'єктах класу **JSpinner** можна використовувати інтерфейс **ChangeListener** і клас **ChangeEvent**.

3.1.7. Текстові компоненти

3.1.7.1. Клас JTextComponent

Батьківським класом текстових компонент в Swing є клас **JTextComponent**. Він забезпечує функціональні можливості, які є загальними для всіх текстових компонент.

Можливості компонента **JTextComponent** бібліотеки Swing набагато перевершують вимоги, що пред'являються до звичайного текстового поля або напису. Даний компонент предоставляет розробникам великий набір методів роботи з текстом, наприклад:

- **`public void copy ();`** - копіювати в буфер обміну;
- **`public void cut ();`** - вирізати в буфер обміну;
- **`public void paste ();`** - вставити з буфера обміну;
- **`public String getSelectedText ();`** - отримати виділений текст;
- **`public int getSelectionStart ();`** - визначити точку початку виділеного тексту;
- **`public int getSelectionEnd ();`** - визначити точку кінця виділеного тексту;
- **`public String getText ();`** - ввести текст з поля;
- **`public void setText (String text);`** - помістити текст в поле;
- **`public void setEditable (boolean b);`** - дозволити редагування;
- **`public void setCaretPosition (int position);`** - помістити курсор в зазначену позицію.
-

3.1.7.2. Класи JTextField, JPasswordField і JTextArea

Підкласи класу **JTextComponent** - **JTextField** і **JTextArea** практично аналогічні класам **TextField** і **TextArea** пакету AWT. Між ними існують лише такі відмінності:

1. Рядки введення паролів задаються в Swing за допомогою окремого класу **JPasswordField.** Конструктори цього класу аналогічні конструкторам класу **JTextField**. Метод

`public char [] getPassword ();`

дозволяє отримати значення введеного пароля, а методи

`public void setEchoChar (char c);`

і

`public char getEchoChar ();`

- встановити і отримати значення луна-символу;

2. Прокрутка об'єкта класу **JTextArea** (так само, як і об'єкта класу **JList**) виконується за допомогою приміщення списку в об'єкт класу **JScrollPane**.

Для обробки подій в об'єктах класу **JButton** можна використовувати інтерфейс **ActionListener** і клас **ActionEvent**. Однак зазвичай з текстовими компонентами пов'язана кнопка, для якої і виконується обробка події.

3.1.8. Меню

Компоненти **JMenuBar**, **JMenu**, **JMenuItem** і **JCheckBoxMenuItem** в Swing за своїми функціональними можливостями і використанню аналогічні компонентам **MenuBar**, **Menu**, **MenuItem** і **CheckboxMenuItem**, використовуваним в AWT.

Новий компонент **JRadioButtonMenuItem** дозволяє організувати в меню перемикачі. Група альтернативних перемикачів формується таким чином: спочатку створюється об'єкт класу **ButtonGroup**, а потім в цей об'єкт за допомогою методу `add ();` додаються об'єкти класу **JRadioButtonMenuItem**.

Класи, що реалізують меню в Swing, відрізняються від відповідних класів AWT наступними основними особливостями:

• оскільки всі класи, крім **JMenuBar**, є підкласами класу **JAbsractButton**, для меню і пунктів меню можна задавати не тільки текст, а й зображення (для цього в конструктори цих класів введено параметр **Icon image** і додані методи, пов'язані з зображеннями);

• рядок меню встановлюється за допомогою методу `public void setJMenuBar (JMenuBar menubar);` класу **JFrame**;

• доданий інтерфейс **MenuListener** для обробки подій пов'язаних з меню (об'єктами класу **JMenu**). Методи цього інтерфейсу

```
public void menuSelected (MenuEvent e);
public void menuDeselected (MenuEvent e);
public void menuCanceled (MenuEvent e);
```

дозволяють обробляти події, пов'язані з вибором меню, скасуванням вибору меню і скасуванням меню. Блоки прослуховування для меню додаються і видаляються за допомогою методів

```
public void addMenuListener (MenuListener l)
```

і

```
public void removeMenuListener (MenuListener l);
```

мнемонічні клавіші («гарячі» клавіші) для меню і пунктів меню на відміну від AWT діють тільки тоді, коли відповідні меню або пункти меню виведені на екрані. Для того, щоб клавіші діяли незалежно від того, чи видно відповідний елемент меню на екрані, необхідно задати такі клавіші за допомогою методу `public void setAccelerator (KeyStroke keyStroke);`

• для пунктів меню введений інтерфейс **MenuDragMouseListener**, методи якого

```
public void menuDragMouseEntered (MenuDragMouseEvent e);
public void menuDragMouseExited (MenuDragMouseEvent e);
public void menuDragMouseDragged (MenuDragMouseEvent e);
public void menuDragMouseReleased (MenuDragMouseEvent
e);
```

дозволяють організувати обробку подій, пов'язаних з переміщенням курсора миші над пунктами меню (наприклад, зміни виду пункту меню при наведенні на нього курсора миші). Блоки прослуховування для меню додаються і видаляються за допомогою методів

```
public void addMenuDragMouseListener (
MenuDragMouseListener l);
```

і

```
public void removeMenuDragMouseListener (
MenuDragMouseListener l);
```

Крім цього, для пунктів меню введений інтерфейс **MenuKeyListener**, методи якого

```
public void menuKeyTyped (MenuKeyEvent e);
public void menuKeyPressed (MenuKeyEvent e);
public void menuKeyReleased (MenuKeyEvent e);
```

дозволяють організувати обробку подій, пов'язаних з натисканням і відпусканням, а також окремо з натисканням і окремо з відпуском клавіш, пов'язаних з пунктами меню.

3.1.9. Бігунки і підказки

3.1.9.1. Класи JScrollBar і JSlider

Смуга прокрутки (клас **JScrollBar**) в Swing практично не відрізняється від смуги прокрутки (клас **Scrollbar**) в AWT.

Бігунки реалізуються в Swing за допомогою класу **JSlider**. Цей компонент, як і **JScrollBar**, дозволяє вибрати значення за допомогою переміщення бігунка. Основними конструкторами класу **JSlider** є

```
JSlider ();
JSlider (int orientation);
JSlider (int min, int max);
JSlider (int min, int max, int value);
JSlider (int orientation, int min, int max, int value);
```

де параметр **orientation** задає орієнтацію, а параметри **min**, **max** і **value** відповідно мінімальне, максимальне і поточне значення бігунка.

Для бігунків можна задати лінійки з ризиками великий або маленької висоти з використанням методів

```
public void setMajorTickSpacing (int n);
```

і

```
public void setMinorTickSpacing (int n);
```

де параметр ***n*** задає відстань між ризиками в пікселях.

Для відстеження подій бігунка використовується інтерфейс **ChangeListener** і клас **ChangeEvent**.

3.1.9.2. Клас JToolTip

«Спливаючі» підказки (текст, що з'являється, якщо навести на деякий час покажчик миші на будь-який об'єкт) реалізуються в Swing за допомогою класу **JToolTip**. Об'єкт цього класу створюється за допомогою конструктора

```
JToolTip ();
```

Прикріпити підказки до компоненту і отримати компонент, до якого прикріплена дана підказка, можна за допомогою методів

```
void setComponent (JComponent comp);
```

і

```
JComponent getComponent ();
```

а установка і отримання тексту підказки виконується за допомогою методів

```
void setTipText (String tipText);
```

і

```
String getTipText ();
```

Методи класу **JComponent**

```
public void setToolTipText (String text);
```

і

```
public String getToolTipText ();
```

дозволяють встановити і отримати підказку безпосередньо для будь-якого графічного компонента.

3.1.10. Оформлення рамок

Кожен стандартний графічний компонент в Java міститься в прямокутній області на екрані зі сторонами, паралельними сторонам екрана. Для деяких графічних компонент сторони цього прямокутника виділяються певним чином. Так, наприклад, сторони кнопки класу **JButton** намальовані так, що створюють враження її опуклості. При натисканні кнопки миші на ній графічне оформлення сторін кнопки **JButton** змінюється, створюючи враження її «вдавленности».

Бібліотека Swing дозволяє змінити оформлення границь будь-якого компонента, в тому числі і контейнера за допомогою рамок різного виду.

Найзагальніші характеристики всіх рамок описані інтерфейсом **Border** з пакета **javax.swing.border**. Креслення рамки виконується методом

```
public void paintBorder (Component c, Graphics g,
int x, int y, int width, int height);
```

Тут задається компонент ***c***, який обводиться рамкою, екземпляр ***g*** класу **Graphics**, що володіє методами малювання, і розміри рамки, які зазвичай збігаються з розмірами компонента, або трохи більше або трохи менше їх.

Рамка може бути прозорою або непрозорою. Це зазначається логічним методом

```
public boolean isBorderOpaque ();
```

Третій і останній метод інтерфейсу

```
public Insets getBorderInsets (Component c);
```

повертає простір, зайнятий рамкою даного компонента ***c***, у вигляді екземпляра класу **Insets**. У класі **Insets** цей простір визначається товщиною рамки зверху **top**, зліва **left**, справа **right** і знизу **bottom**. Всі чотири поля класу **Insets** є цілочисельними переменнимі.

Клас **AbstractBorder** розширюють близько двадцяти класів, що викреслюють найрізноманітніші рамки. Для зручності роботи з ними в пакеті **javax.swing** є клас **BorderFactory**, в якому зібрані статичні методи виду **createXxxBorder ()** для різних типів рамок з різними параметрамі. Найчастіше для створення рамки досить скористатися одним з цих методів, а потім встановити отриману рамку в компонент методом

```
public void setBorder (Border border);
```

класу **JComponent**, наприклад:

```
JLabel myLabel = new JLabel ("Напис в товстої
синьої рамкою");
myLabel.setBorder (BorderFactory.createLineBorder (
Color.blue, 3));
```

Нижче наведені основні статичні методи класу **BorderFactory** для створення рамок різних типів:

```
public static Border createEmptyBorder ();
```

створює порожню рамку з нульовими розмірами.

```
public static Border createEmptyBorder (int top, int
left, int bottom, int right);
```

створює порожню рамку з заданими розмірами.

```
public static Border createLineBorder (Color color);
public static Border createLineBorder (Color color, int
thickness);
```

створюють відповідно рамку з заданим кольором і рамку з заданим кольором заданої товщини.

```
public static Border createBevelBorder (int type);
```

створює рамку заданого типу **type** (опуклого - `BevelBorder.RAISED` або увігнутого - `BevelBorder.LOWERED`).

```
public static Border createBevelBorder (int type, Color
highlight, Color shadow);
```

створює рамку заданого типу з заданим світлим (highlight) і темним (shadow) кольором.

```
public static Border createBevelBorder (int type, Color
highlightOuter, Color highlightInner, Color shadowOuter,
Color shadowInner);
```

створює об'ємну двобарвну рамку заданого типу (внутрішні лінії мають кольору **highlightInner** і **shadowInner**, а зовнішні - кольори **highlightOuter** і **shadowOuter**);

```
public static Border createEtchedBorder ();
```

створює стандартну «врізану» рамку з кольорами трохи світліше і трохи темніше кольору фону.

```
public static Border createEtchedBorder (int type);
```

створює «врізану» рамку (якщо **type** дорівнює **EtchedBorder.RAISED**) або «вдавлену» (якщо **type** дорівнює **EtchedBorder.LOWERED**).

```
public static Border createEtchedBorder (Color highlight, Color shadow);
```

задає кольори «врізаної рамки».

```
public static Border createEtchedBorder (int type, Color highlight, Color shadow);
```

об'єднує можливості двох попередніх методів.

Рамка одного кольору, але з ліній різної товщини задається за допомогою методу

```
public static MatteBorder createMatteBorder (int top, int left, int bottom, int right, Color color);
```

де **color** - колір рамки, **top**, **left**, **bottom** і **right** - товщина рамки (в пікселях) відповідно зверху, зліва, знизу і праворуч.

```
public static MatteBorder createMatteBorder (int top, int left, int bottom, int right, Icon tileIcon);
```

Метод, що задає як рамки повторюване зображення **tileIcon**.

```
public static TitledBorder createTitledBorder (Border border, String title, int titleJustification, int titlePosition, Font titleFont, Color titleColor);
```

задає для рамки **border** напис **title** шрифтом **titleFont** і кольору **titleColor**, вирівняний по горизонталі відповідно до значення **titleJustification** (одна з статичних констант класу **TitledBorder**: **LEFT**, **CENTER**, **RIGHT**, **LEADING**, **TRAILING** або **DEFAULT_JUSTIFICATION**, - значення за замовчуванням **LEADING**) і по вертикалі відповідно до значення **titlePosition** (одна з статичних констант класу **TitledBorder**: **ABOVE_TOP**, **TOP**, **BELOW_TOP**, **ABOVE_BOTTOM**, **BOTTOM**, **BELOW_BOTTOM** або **DEFAULT_POSITION** - значення за замовчуванням **TOP**) .

Решта (п'ять) методів

createTitledBorder () містять окремі параметри або частину параметрів наведеного вище методу;

```
public static CompoundBorder createCompoundBorder (Border outsideBorder, Border insideBorder);
```

використовується для створення рамки, що складається з двох вкладених рамок будь-яких типів.

Хоча бібліотека Swing надає безліч готових рамок і клас **BorderFactory** для їх швидкого створення, часто виникає необхідність сконструювати оригінальну рамку. Для її створення можна розширити абстрактний клас **AbstractBorder**, визначивши хоча б один конструктор, і перевизначивши методи **paintBorder ()** і **getBorderInsets ()**. Якщо рамка непрозора, то треба перевизначити метод **isBorderOpaque ()** так, щоб він повертав ***true***.

Свою рамку можна створити, розширивши який-небудь клас рамок, наприклад, для рамки, що розширює клас **TitledBorder**, в заголовок можна вставити не напис, а компонент класу **JComponent**.

3.2. Основні менеджери компонування Java

3.2.1. Принципи функціонування менеджерів компонування

При додаванні компонентів в контейнер зазвичай не потрібно вказувати положення кожного компонента в межах контейнера. За допомогою менеджерів компонування можна вказати, як ті чи інші компоненти повинні бути розташовані по відношенню до інших компонентів. Точні значення координат обчислює сам менеджер. Таким чином, полегшується побудова програм, що не залежать від використовуваного Web-браузера та розміру дисплея.

Інтерфейс **LayoutManager** є інтерфейсом в бібліотеці класів Java, що описує як клас **Container** і менеджер компонування взаємодіють між собою. Він оголошує кілька методів.

Метод, що розташовує контейнер в заданій панелі:

```
public void layoutContainer (Container parent);
```

Методи, що інформують менеджер компонування про додавання і видалення компонентів з контейнера:

```
public void addLayoutComponent (String name, Component
comp);
public void removeLayoutComponent (Component comp);
```

Методи, що задають розміри компонентів, якими вони управляють:

```
public Dimension minimumLayoutSize (Container parent);
public Dimension preferredLayoutSize (Container parent);
```

Додатковий інтерфейс з методами позиціонування і перевірки, **LayoutManager2**, був доданий в JDK 1.1. У цьому інтерфейсі оголошений метод

```
public Dimension maximumLayoutSize (Container parent);
```

і методи, які повертають вирівнювання по осях ***x*** і ***y***:

```
public float getLayoutAlignmentX (Container target);
public float getLayoutAlignmentY (Container target);
```

Крім того, метод, що додає заданий компонент з використанням заданого об'єкта-обмежувача:

```
public void addLayoutComponent (Component comp,
Object constraints);
```

Метод, що робить контейнер недійсним:

```
public void invalidateLayout (Container target);
```

В AWT визначені п'ять різних типів менеджерів компонування:

- **FlowLayout** - послідовне розташування;
- **GridLayout** - табличне розташування;
- **BorderLayout** - полярне розташування;

• **CardLayout** - блокнотне розташування;

• **GridBagLayout** - пористе розташування.

Всі ці менеджери реалізують інтерфейс **LayoutManager**, а менеджери **BorderLayout**, **CardLayout** і **GridBagLayout** - також інтерфейс **LayoutManager2**. Установка менеджера компоновки для контейнера виконується за допомогою методу

```
public void setLayout (LayoutManager manager);
```

де параметр **manager** визначає об'єкт одного з класів менеджерів компонування, наприклад:

```
setLayout (new GridLayout (3,2));
```

Метод, що повертає менеджер компонування для даного контейнера:

```
public LayoutManager getLayout ();
```

Для визначення проміжків між границею контейнера і компонент, що містяться в ньому, використовується клас **Insets**. В даному класі містяться властивості

public int top,
public int left,
public int bottom,
public int right,

що задають відступ зверху, праворуч, знизу і зліва (в пікселях). Менеджер компонування створює об'єкт класу **Insets**, звернувшись до конструктору класу

```
public Insets (int top, int left, int bottom, int right);
```

який повертає об'єкт **Insets**. Параметри **top**, **left**, **bottom** і **right** визначають проміжки (в пікселях) відповідно від верхньої, лівої, нижньої і правої меж контейнера. Наприклад, якщо потрібно залишити простір шириною 20 пікселів знизу і зверху і 10 пікселів по боках, потрібно визначити в програмі наступний метод **insets ():**

```
public Insets insets ()
{
return new Insets (20, 10, 20, 10);
}
```

Для виконання своєї роботи менеджер компонування повинен зв'язуватися з об'єктом класу **Container**. Якщо контейнер не має пов'язаного з ним менеджера компоновки, він просто поміщає компоненти з використанням методів класу **Component:**

Метод, що пересуває і змінює розміри компонента так, щоб він вписувався в заданий прямокутник:

```
public void setBounds (Rectangle r);
```

Метод, що переміщує компонент:

```
public void setLocation (Point p);
```

Метод, що змінює розміри компонента:

```
public void setSize (Dimension d);
```

Якщо контейнер має пов'язаний з ним менеджер компонування, контейнер запитує менеджер компонування про позиціонування і розміри своїх компонент перед тим, як вони будуть намальовані. Сам менеджер компонування не виконує малювання, він просто вирішує, який обсяг і позицію повинен мати кожен компонент і викликає методи **`setBounds ()`**, **`setLocation ()`** і / або **`setSize ()`** для кожного з цих компонентів.

Як вже зазначалося вище, **LayoutManager** пов'язаний з об'єктом класу **Container** викликом методу **`setLayout ()`**, наприклад:

```
Panel p = new Panel ();
p.setLayout (new BorderLayout ());
```

Деякі контейнери, такі як **Panel**, забезпечує конструктор, який в якості аргументу сприймає менеджер компонування:

```
Panel p = new Panel (new BorderLayout ());
```

При створенні власних менеджерів компонування також повинен бути реалізований інтерфейс **LayoutManager** і, можливо, **LayoutManager2**.

Власні менеджери розташування зазвичай використовуються в тих випадках, коли можливості, що надаються стандартними менеджерами, є недостатніми.

У тих випадках, коли потрібно вручну розташувати компоненти, використовується абсолютне позиціонування компонент. Оскільки навіть коли менеджер компонування не заданий, діє менеджер за замовчуванням (**FlowLayout** для **Panel** і **BorderLayout** для **Frame**), необхідно відключити менеджери компонування, що діють за умовчанням, за допомогою оператора

```
setLayout (null);
```

3.2.2. Менеджер FlowLayout

Клас **FlowLayout** розглядає як контейнер набір рядків. Цей менеджер використовується за умовчанням в об'єктах класів **Panel** і **Applet**.

Це найпростіший з менеджерів компонування AWT. Його стратегія компонування є наступною:

- враховувати бажаний розмір всіх розміщених компонент.
- скомпонувати якомога більше компонент горизонтально всередині контейнера.
- почати новий рядок компонент, якщо вони ще є.
- якщо всі компоненти не відповідають розмірам, то, компоненти, які не поміщаються, що не виводяться.

Висота кожного рядка визначається по висоті компонентів, що знаходяться в ній. Менеджер **FlowLayout** додає компоненти зліва направо. Якщо наступний компонент не поміщається в поточному рядку, відбувається перехід на новий рядок, який знову починається зліва. Можна розміщувати рядки з вирівнюванням вліво, вправо і по центру. За замовчуванням використовується вирівнювання по центру.

конструктор

```
public FlowLayout (int alignment, int hgap, int vgap);
```

створює об'єкт **FlowLayout**. Параметр **alignment** регулює вирівнювання рядків, а можливі типи вирівнювання задаються наступними змінними класу **FlowLayout** з модифікаторами **public static final int**: **LEFT** (вліво), **RIGHT** (вправо) і **CENTER** (по центру). Параметри **hgap** і **vgap** задають розміри проміжків між компонентами по вертікалі і горизонталі (в пікселях). Так, щоб розташувати компоненти з правим вирівнюванням з проміжком 10 пікселів по вертикалі і 5 пікселів по горизонталі, треба створити об'єкт **FlowLayout** наступним чином:

```
myFlowLayout = new FlowLayout (FlowLayout.RIGHT, 10, 5);
```

Можна також використовувати конструктори

```
public FlowLayout (int alignment)
```

і

```
public FlowLayout ()
```

За замовчуванням **hgap** і **vgap** рівні 5 пікселів і вирівнювання проводиться по центру.

Кращим для **FlowLayout** буде розміщення всіх його компонент в одному ряду. Це означає, що його краща висота повинна бути максимальною висотою компонент плюс проміжок, рівний **vgap**. Бажана ширина повинна бути сумою всіх значень ширини містяться компонент плюс відстані між компонентами плюс відстані між компонентом і межами ряду, рівні **hgap** (**vgap** і **hgap** є загальними властивостями більшості стандартних менеджерів компонування і визначають, як далеко компоненти розміщуються один від одного).

Клас **FlowLayout**, крім методів інтерфейсу **LayoutManager**, забезпечує також наступні методи:

Отримання та встановлення вирівнювання:

```
public int getAlignment ();
public void setAlignment (int align);
```

Отримання та встановлення проміжку по вертикалі між компонентами:

```
public int getVgap ();
public void setVgap (int vgap);
```

Отримання та встановлення проміжку по горизонталі між компонентами:

```
public int getHgap ();
public void setHgap (int hgap);
```

Менеджер **FlowLayout** орієнтований по горизонталі. Якщо необхідна вертикальна компоновка, потрібно написати свій власний менеджер (або використовувати менеджер **BoxLayout** з бібліотеки Swing).

3.2.3. Менеджер GridLayout

Клас **GridLayout** розділяє контейнер на клітини однакового розміру (по типу таблиці). При додаванні компонентів в контейнер **GridLayout** конструктор

```
public GridLayout (int rows, int cols, int hgap, int vgap);
```

розміщує їх в клітинах зліва направо і зверху вниз. Параметри **rows** і **cols** задають кількість рядків або стовпців таблиці, **hgap** а й **vgap** - відповідно проміжки між стовпцями і рядками. Якщо задано кількість рядків, кількість стовпців буде обчислено. Якщо, навпаки, поставити кількість стовпців, обчислюватися буде кількість рядків. Якщо додати шість компонентів в об'єкт **GridLayout**, що містить два рядки, він створить три стовпці. Якщо створюється **GridLayout** із заданим числом рядків, слід як числа стовпців вказати значення 0. Якщо ж треба задати число стовпців, то замість числа рядків слід задати значення 0 (якщо обидва значення ненульові, буде використовуватися тільки кількість рядків, а число стовпців буде обчислено, виходячи з кількості рядків і числа компонентів в контейнері).

Наприклад, рядок

```
GridLayout myGridLayout = new GridLayout (0, 4, 8, 10);
```

створює об'єкт **GridLayout** з чотирма стовпцями, промежутком між стовпцями 8 пікселів, а між рядками - 10 пікселів. Завдання нульових значень і для **rows** і для **cols** призводить до порушення виключення **IllegalArgumentException**.

Можна також створити об'єкт **GridLayout** за допомогою конструктора

```
public GridLayout (int rows, int cols);
```

(В цьому випадку проміжків між стовпцями і між рядками не буде).

Конструктор

```
public GridLayout ();
```

задає об'єкт з одним рядком і однієї колонкою.

Менеджер **GridLayout** намагається встановити розмір кожного компонента до максимальної бажаної ширині і максимальної бажаної висоті (всі компоненти **GridLayout** будуть одного і того ж розміру). Тому бажана ширина **pw** і висота **ph** для **GridLayout** розраховується за такою формулою:

```
pw = (maxPrefWidth * cols) + (hgap * (cols + 1));
ph = (maxPrefHeight * rows) + (vgap * (rows + 1));
```

Клас **GridLayout**, крім методів інтерфейсу **LayoutManager**, забезпечує також наступні методи:

Отримання та встановлення вирівнювання:

```
public int getAlignment () ;
public void setAlignment (int align);
```

Отримання та встановлення проміжку по вертикалі між компонентами:

```
public int getVgap ();
public void setVgap (int vgap);
```

Отримання та встановлення проміжку по горизонталі між компонентами:

```
public int getHgap ();
public void setHgap (int hgap);
```

Отримання та встановлення кількості рядків:

```
public int getRows ();
public void setRows (int rows);
```

Отримання та встановлення кількості стовпців:

```
public int getColumns ();
public void setColumns (int rows);
```

3.2.4. Менеджер BorderLayout

Клас **BorderLayout** розділяє контейнер на п'ять областей, умовно названих "**North**" (північ), "**South**" (південь), "**East**" (схід), "**West**" (захід) і "**Center**" (центр). Цей менеджер використовується за умовчанням в об'єктах класу **Frame** і його підкласах. Насправді з п'яти наведених областей можна використовувати, в залежності від завдання, дві або три області.

Менеджер **BorderLayout** вимагає обмежувача при додаванні компонент. Обмежувач може бути однією з наступних змінних класу **BorderLayout** з модифікаторами **public static final int**: **NORTH**, **SOUTH**, **EAST**, **WEST** або **CENTER**.

Ці обмежувачі визначаються за допомогою наступного методу класу Container:

```
public void add (Component component, Object constraint);
```

Наприклад:

```
add (new Button ( "Північ"), BorderLayout.NORTH);
```

Об'єкт BorderLayout створюється за допомогою одного з конструкторів:

```
public BorderLayout ();
public BorderLayout (int hgap, int vgap);
```

де параметри **vgap** і **hgap** задають інтервали по вертикалі і горізонталі (в разі конструктора без параметрів між компонентами немає проміжків).

Клас **BorderLayout** не дозволяє додати в дану область більш одного компонента. Якщо спробувати додати два компоненти в одну і ту ж область, видно буде лише той з них, який був доданий останнім.

Менеджер діє за наступним алгоритмом:

- області **BorderLayout.NORTH** і **BorderLayout.SOUTH** є корисними, якщо необхідно пов'язати висоту компонента з його бажаною висотою;
- області **BorderLayout.EAST** і **BorderLayout.WEST** є корисними, якщо необхідно пов'язати ширину компонента з його бажаною шириною;
- як тільки зазначені вище компоненти стають пов'язаними, область **BorderLayout.CENTER** стає частиною, що розширюється.

Бажана ширина **pw** компонування **BorderLayout** залежить від найширшої з рядків, а бажана висота **ph** залежить від розмірів компонент **NORTH** і **SOUTH**, а також найвищого з компонент середнього рядку:

```
pw = max (north.pw, south.pw,
(West.pw + center.pw + east.pw + hgaps));
ph = vgaps + north.ph + south.ph +
max (west.ph, center.ph, east.ph);
```

Клас **BorderLayout**, крім методів інтерфейсів **LayoutManager** і **LayoutManager2**, забезпечує також наступні методи:

Отримання та встановлення проміжку по вертикалі між компонентами:

```
public int getVgap ();
public void setVgap (int vgap);
```

Отримання та встановлення проміжку по горизонталі між компонентами

```
public int getHgap ();
public void setHgap (int hgap);.
```

3.2.5. Менеджер компонування BoxLayout

Компонування **BoxLayout** в цілому подібне до стандартного компонуванні бібліотеки AWT **FlowLayout**, за винятком того, що при роботі з новою компоновкою можна визначити використовувану вісь за допомогою статичних змінних класу **BoxLayout**: **X_AXIS** або **Y_AXIS**. На відміну від компонування **GridLayout**, тут кожен компонент може займати осередок різного розміру. Для використання компонування **BoxLayout** з орієнтацією по осі ***Y*** досить помістити в програму наступний оператор:

```
setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
```

Якщо в програмі бажано застосувати компоновку типу **BoxLayout**, як контейнер замість класу **JPanel** слід використовувати клас **Box** (для цього класу компоновка **BoxLayout** використовується за замовчуванням). Крім того, клас **Box** надає кілька додаткових методів, що забезпечують повний контроль над розміщенням компонентів у вікні.

Так, до числа розміщуваних на панелі компонентів можна включити невидимі компоненти-наповнювачі трьох видів.

Перший вид заповнювача - невидима розділова область (**rigid area**), що має фіксовані розміри. Вона створюється за допомогою методу

```
public static Component createRigidArea (Dimension d);
```

і вставляється між компонентами, створюючи проміжок фіксованого розміру між ними.

Заповнювач другого виду - невидима «розпірка» (**strut**) має тільки один фіксований розмір. У горизонтальній розпірки, створюваної статичним методом

```
public static Component createHorizontalStrut (int width);
```

фіксована тільки ширина. При горизонтальному розташуванні компонентів розпірку можна використовувати для створення проміжків між компонентами, а при вертикальному розташуванні - для завдання ширини всієї панелі.

У вертикальній розпірці, створюваної статичним методом

```
public static Component createVerticalStrut (int height)
```

фіксована висота. Вона використовується аналогічно горизонтальній розпірці.

Третій вид заповнювача - невидима «надувна подушка» (**glue**), «роздуваючи», заповнює весь виділений їй простір, розсовуючи інші компоненти і притискаючи їх до країв панелі, якщо вони мають фіксований максимальний розмір. Цей заповнювач створюється одним з статичних методів:

```
public static Component createGlue ();
```

- «подушка» роздувається на всі боки;

```
public static Component createHorizontalGlue ();
```

- «подушка» роздувається в ширину;

```
public static Component createVerticalGlue ();
```

- «подушка» роздувається в висоту.

Крім цих трьох компонентів-роздільників можна використовувати невидимий компонент з фіксованим мінімальним, максимальним і бажаним розмірами. Він є об'єктом класу **Filler**, вкладеного в клас **Box**, і створюється конструктором

```
Box.Filler (Dimension min, Dimension pref, Dimension max);
```

Перевага цього об'єкта в тому, що він може поміняти розміри за допомогою методу

```
void changeShape (Dimension min, Dimension pref,
Dimension max);
```

3.2.6. Менеджер компонування SpringLayout

Менеджер компонування **SpringLayout** використовує для визначення положення і розмірів компонента координати ***x***, ***y*** і розміри **width**, **height**, що задаються об'єктами абстрактного класу **Spring**.

Кожен об'єкт цього класу містить чотири величини: мінімальний, максимальний та бажаний розміри, а також поточний розмір. Ці розміри можуть використовуватися як розміри проміжків між компонентами. Об'єкт класу **Spring** задається методом із заданим мінімальним, бажаним і максимальним значеннями:

```
public static Spring constant (int min, int pref, int max);
```

Метод, що повертає об'єкт класу **Spring** з співпадаючими між собою розмірами, рівними **size**:

```
public static Spring constant (int size)
```

Поточний розмір value, встановлюється методом

```
public abstract void setValue (int value).
```

Мінімальний, максимальний, бажаний і поточний розмір можна отримати за допомогою методів

```
public abstract int getMinimumValue ();
public abstract int getMaximumValue ();
public abstract int getPreferredValue ();
public abstract int getValue ();
```

При використанні менеджером **SpringLayout** декількох об'єктів класу для обліку їх взаємодії в класі **Spring** визначені кілька методів.

Так, метод, що повертає новий об'єкт, розміри якого складені з найбільших значень об'єктів **s1** і **s2**:

```
public static Spring max (Spring s1, Spring s2);
```

Метод, що повертає новий об'єкт, розміри якого дорівнюють розмірам об'єкту s з протилежним знаком:

```
public static Spring minus (Spring s)
```

Метод, що повертає новий об'єкт, розміри якого дорівнюють сумі відповідних розмірів об'єктов s1 і s2:

```
public static Spring sum (Spring s1, Spring s2);
```

Об'єкт класу **Spring** використовується для побудови об'єкта вкладеного класу **SpringLayout.Constraints**.

Об'єкт цього класу містить координати ***x*** і ***y***, ширину **width** і висоту **height**, що є об'єктами класу **Spring**. Менеджер розміщення **SpringLayout** змінює положення лівого верхнього кута компонента в заданих об'єктами ***x*** і ***y*** межах. Тому координати позначаються статичними строковими константами **WEST** і **NORTH** класу **SpringLayout**. Величини ***x* + *width*** і ***y* + *height*** позначаються статичними строковими константами **EAST** і **SOUTH** і визначають положення правого нижнього кута компонента.

Для отримання і установки всіх цих значень в класі **SpringLayout.Constraints** визначені методи:

Установка і отримання координати ***x***:

```
void setX (Spring x);
Spring getX ();
```

Установка і отримання координати ***y***:

```
void setY (Spring y);
Spring getY ();
```

Установка і отримання ширини:

```
void setWidth (Spring width);
Spring getWidth ();
```

Установка і отримання висоти:

```
void setHeight (Spring height);
Spring getHeight ();
```

Методи, що встановлюють і видають об'єкт класу **Spring** по заданому імені **NORTH**, **WEST**, **SOUTH** або **EAST**

```
public void setConstraint (String edgeName, Spring s);
public Spring getConstraint (String edgeName);
```

3.2.7. Менеджер компонування GroupLayout

Менеджер **GroupLayout** ієрархічно групує компоненти для їх позиціонування в контейнері і підтримує два типи груп. Групи першого типу вибудовують свої елементи послідовно, групи другого типу - паралельно чотирма різними способами. Групування виконується за допомогою об'єктів класу **Group**.

Кожна група може містити будь-яку кількість об'єктів класу **Group** або **Component**, а також проміжок (**gap**). Компоненти менеджера схожі на

пружини. Кожен компонент має діапазон розмірів, що задається мінімальним, бажаним і максимальним розмірами, що визначаються за допомогою методів **`getMinimumSize ()`**, **`getPreferredSize ()`** і **`getMaximumSize ()`** класу **Component**. Проміжок також розглядається як невидимий компонент з мінімальним, бажаним і максимальним розміром. Крім того, менеджер підтримує бажаний проміжок, який можна отримати за допомогою методу **`getPreferredGap ()`** класу **LayoutStyle**.

Діапазон для об'єкта класу визначається типом групи: для послідовних груп діапазон дорівнює сумі елементів, для паралельних - діапазону для максимального елемента.

Менеджер **GroupLayout** розглядає кожну вісь незалежно, тобто існує група, що представляє вертикальну вісь і група, що представляє горизонтальну вісь. Вертикальна вісь визначає мінімальний, бажаний і максимальний розмір по цій осі, установку координати ***y*** і висоти компонент, а горизонтальна вісь - мінімальний, бажаний і максимальний розмір по цій осі, установку координати ***x*** і ширини компонент. Кожен компонент повинен бути заданий і у вертикальній і в горизонтальній групі.

Конструктор, що задає групову компонування для контейнера **host:**

```
public GroupLayout (Container host);
```

Метод, що задає (при значенні параметра **autoCreatePadding**, рівному ***true***) або скасовує (при значенні параметра **autoCreatePadding**, рівному ***false***) автоматичну вставку проміжків між компонентами:

```
public void setAutoCreateGaps (boolean
autoCreatePadding);
```

Горизонтальна і вертикальна групи створюються за допомогою методів

```
public void setHorizontalGroup (
GroupLayout.Group group);
public void setVerticalGroup (GroupLayout.Group group);
```

Обидва ці методи містять в якості параметра об'єкт group вкладеного класу Group, який задає позиції і розміри компонент вздовж горизонтальної або вертикальної осі.

Послідовна або паралельна групи створюються за допомогою методів

```
public GroupLayout.SequentialGroup
createSequentialGroup ();
public GroupLayout.ParallelGroup createParallelGroup ();
public GroupLayout.ParallelGroup createParallelGroup (
GroupLayout.Alignment alignment);
GroupLayout.ParallelGroup createParallelGroup (
GroupLayout.Alignment alignment, boolean resizable);
```

Передостанній і останній методи задають (за допомогою параметра **alignment**) вирівнювання компонент в групі за допомогою перелічуваних змінних класу **GroupLayout.Alignment**: **LEADING** (вирівнювання до початку групи), **TRAILING** (вирівнювання до кінця групи), **CENTER** (вирівнювання по центру) і **BASELINE** (вирівнювання за базовою лінією).

Останній метод задає також (за допомогою параметра **resizable**) можливість зміни розміру компонента.

Методи

```
public void linkSize (Component ... components);
public void linkSize (int axis, Component ... components);
```

дозволяють встановити однаковий розмір для всіх заданих як параметри компонент групи (другий метод - тільки для заданої осі, яка задається статичними константами **HORIZONTAL** і **VERTICAL** класу **SwingConstants**).

3.3 Обробка подій в Java

3.3.1. Модель делегування подій в Java

Операційна система, наприклад Microsoft Windows, відстежує всі події, що відбуваються в системі. Система направляє ці події до відповідних цільових об'єкти. Якщо, наприклад, користувач клацає по вікну аплету, система створює подію **mouse-down** (натискання клавіші миші) і відсилає його даним вікна для обробки. Потім це вікно може виконати деяку операцію або просто передати подію назад системі для обробки за замовчуванням.

У мові Java події є об'єктами деяких класів.

В JDK 1.0 існував єдиний клас **Event**, що описує всі події, на які програма може реагувати, а також визначає методи за замовчуванням для отримання інформації про конкретну подію.

Починаючи з JDK 1.1, в Java були змінена модель обміну інформацією про події, названа моделлю делегування подій. Стара модель і клас **Event** були збережені для можливості використання старих програм, однак використовувати стару модель не рекомендується.

Модель делегування подій працює наступним чином. Кожен елемент взаємодії між інтерфейсом користувача і програмою визначається як подія. Класи додатків з'ясовують свою зацікавленість в деякому очікуваному певним компонентом подію шляхом опитування компонента і видачі йому пропозиції помістити в список відомості про блок прослуховування даного компонента (**listener**). Коли відбувається деяка подія, джерело події повідомляє про нього зареєстровані блоки прослуховування.

Події, як і виключення, утворюють ієрархію класів. Коренем цієї ієрархії є клас **EventObject**, що розширює клас **Object**.

Для класу визначені два методи, що визначають відповідно об'єкт - джерело події і строкове представлення об'єкту-події:

```
public Object getSource ();
public String toString ();
```

Події, пов'язані з графічним інтерфейсом користувача, в свою чергу, утворюють гілку в ієрархії класів подій, коренем якої є клас **AWTEvent**, що розширює клас **EventObject**.

Існують два різних типи подій - **низькорівневі** і **семантичні**.

Низькорівневі події пов'язані з фізичними аспектами інтерфейсу користувача - клацанням миші, натисканням клавіш клавіатури і т.д. Низькорівневі події обробляє клас **ComponentEvent** і його нащадки.

Семантичні події будуються на базі низькорівневих подій. Наприклад, для вибору команди меню може знадобитися розкрити список команд першим клацанням миші, а потім другим клацанням вибрати в ньому необхідну команду. Ця послідовність низькорівневих подій може бути об'єднана в одну семантичну подію. Всі інші події, крім подій гілки **ComponentEvent**, є семантичними (ці компоненти будуть розглянуті далі).

3.3.2. Класи і інтерфейси обробки низькорівневих подій

У кожному класі подій визначені відповідні властивості і методи для обробки подій.

Майже кожен тип події, за винятком **PaintEvent** і **InputEvent**, має пов'язаний з ним інтерфейс або інтерфейси блоків прослуховування, в яких оголошено методи обробки даного типу події. Ці інтерфейси є розширеннями інтерфейсу **EventListener**, де не визначено ніяких змінних і методів.

Розглянемо основні властивості і методи класів низькорівневих подій і відповідних їм блоків прослуховування.

3.3.2.1. Клас ComponentEvent і інтерфейс ComponentListener

Клас **ComponentEvent** обробляє події, пов'язані зі зміною компонента. Причину зміни можна визначити за допомогою наступних статичних **final** змінних класу типу **int**:

COMPONENT_MOVED - компонент перемістився;

COMPONENT_RESIZED - компонент змінив розмір;

COMPONENT_HIDDEN - компонент прибраний з екрану;

COMPONENT_SHOWN - компонент з'явився на екрані;

COMPONENT_FIRST - перший номер у списку ідентифікаторів, використовуваних для подій компонента;

COMPONENT_LAST - останній номер у списку ідентифікаторів, використовуваних для подій компонента.

Метод класу, що повертає компонент - джерело події:

```
ublic Component getComponent ();
```

Інтерфейс **ComponentListener** містить оголошення наступних методів обробки подій компонент:

Обробка події зміни розміру компонента:

```
public void componentResized (ComponentEvent e);
```

Обробка події переміщення компонента:

```
public void componentMoved (ComponentEvent e);
```

Обробка події появи компонента на екрані:

```
public void componentShown (ComponentEvent e);
```

Обробка події видалення компонента з екрану:

```
public void componentHidden (ComponentEvent e);
```

3.3.2.2. Клас ContainerEvent і компонент ContainerListener

Клас **ContainerEvent** обробляє події додавання або видалення компонента і містить наступні статичні **final** змінні типу **int**:

COMPONENT_ADDED - в контейнер доданий компонент;

COMPONENT_REMOVED - з контейнера видалений компонент;

CONTAINER_FIRST - перший номер у списку ідентифікаторів, використовуваних для подій контейнера;

CONTAINER_LAST - останній номер у списку ідентифікаторів, використовуваних для подій контейнера.

Методи класу, що повертають відповідно компонент, на який діє подія і контейнер - джерело події:

```
public Component getChild ();
public Container getContainer ();
```

Інтерфейс ContainerListener містить оголошення наступних методів:

Обробка події додавання компонента в контейнер:

```
public void componentAdded (ContainerEvent e);
```

Обробка події видалення компонента з контейнера:

```
public void componentRemoved (ContainerEvent e);
```

3.3.2.3. Клас FocusEvent і інтерфейс FocusListener

Клас **FocusEvent** обробляє події придбання або втрати фокусу і містить наступні статичні **final** змінні типу **int**:

FOCUS_GAINED - отримання фокусу введення;

FOCUS_LOST - втрата фокусу введення;

FOCUS_FIRST - перший номер у списку ідентифікаторів, використовуваних для подій фокуса;

FOCUS_LAST - останній номер у списку ідентифікаторів, використовуваних для подій фокуса.

Метод класу, що повертає інший компонент, пов'язаний зі зміною фокусу:

```
public Component getOppositeComponent ();
```

Метод, що повертає true, якщо подія, пов'язана зі зміною фокусу, є тимчасовою і false - в протилежному випадку:

```
public boolean isTemporary ();
```

В інтерфейсі FocusListener містяться оголошення двох методів:

Обробка події отримання фокусу:

```
public void focusGained (FocusEvent e);
```

Обробка події втрати фокусу:

```
public void focusLost (FocusEvent e);
```

3.3.2.4. Клас WindowEvent і інтерфейси WindowListener, WindowStateListener і WindowFocusListener

Клас **WindowEvent** обробляє події, пов'язані зі зміною стану вікна і містить наступні статичні **final** змінні типу **int**:

WINDOW_OPENED - вікно відкрилося;
WINDOW_CLOSED - вікно закрилося;
WINDOW_CLOSING - спроба закриття вікна;
WINDOW_ACTIVATED - вікно отримало фокус;
WINDOW_DEACTIVATED - вікно втратило фокус;
WINDOW_ICONIFIED - вікно згорнулося в ярлик;
WINDOW_DEICONIFIED - вікно розгорнулося;
WINDOW_GAINED_FOCUS - вікно отримало фокус;
WINDOW_LOST_FOCUS - вікно втратило фокус;
WINDOW_STATE_CHANGED - будь-яка зміна стану вікна з перерахованих вище;
WINDOW_FIRST - перший номер у списку ідентифікаторів, використовуваних для подій вікна;
WINDOW_LAST - останній номер у списку ідентифікаторів, використовуваних для подій вікна.

Методи класу, що для події **WINDOW_STATE_CHANGED** повертають новий або попередній стан вікна:

```
public int getNewState ();
public int getOldState ();
```

Значення, що повертається можна порівнювати на рівність з наступними статичними цілими **final** змінними класу **Frame**:

NORMAL - нормальний стан вікна;
ICONIFIED - згорнуте вікно;
MAXIMIZED_HORIZ - збільшення до максимального розміру по горизонталі;
MAXIMIZED_VERT - збільшення до максимального розміру по вертикалі;
MAXIMIZED_BOTH - збільшення до максимального розміру по горизонталі і вертикалі.

Методи, що повертають відповідно інше вікно, яке торкнулося дана подія і вікно - джерело події:

```
public Window getOppositeWindow ();
public Window getWindow ();
```

Інтерфейс **WindowListener** містить оголошення наступних методів обробки подій вікна:

Обробка відкриття вікна:

```
public void windowOpened (WindowEvent e);
```

Обробка спроби закриття вікна:

```
public void windowClosing (WindowEvent e);
```

Обробка закриття вікна:

```
public void windowClosed (WindowEvent e);
```

Обробка згортання вікна:

```
public void windowIconified (WindowEvent e);
```

Обробка розгортання вікна:

```
public void windowDeiconified (WindowEvent e);
```

обробка активізації вікна:

```
public void windowActivated (WindowEvent e);
```

Обробка деактивизации вікна:

```
public void windowDeactivated (WindowEvent e);
```

Починаючи з версії 1.4 додані два нових інтерфейси для обробки подій вікна. Інтерфейс **WindowStateListener**, містить оголошення методу, який обробляє подію зміни вікна:

```
public void windowStateChanged (WindowEvent e);
```

Другий інтерфейс **WindowFocusListener** містить оголошення наступних двох методів:

```
void windowGainedFocus (WindowEvent e);
```
- обробка події придбання вікном фокуса;

```
public void windowLostFocus (WindowEvent e);
```
- обробка події втрати вікном фокуса.

3.3.2.5. Клас InputEvent

Клас **InputEvent** є кореневим класом для всіх подій введення рівня компонента. Події введення направляються блокам прослуховування до їх звичайної обробки джерелом, який викликав цю подію. Це дозволяє блокам прослуховування і подклассам компонент «споживати» подію так, щоб джерело не обробляло їх звичайним чином. Наприклад, «споживання» події натискання кнопки запобігає активізації кнопки.

Клас містить наступні статичні **final** змінні типу **int**, які використовуються при обробки подій миші і клавіатури:

BUTTON1_DOWN_MASK, **BUTTON2_DOWN_MASK** і **BUTTON3_DOWN_MASK** - натискання відповідно першої, другої або третьої кнопки миші;

CTRL_DOWN_MASK, **ALT_DOWN_MASK** і **SHIFT_DOWN_MASK** - натискання відповідно клавіш **Ctrl**, **Alt** або **Shift**.

Метод класу, що «споживає» подію так, щоб вона не оброблялася звичайним чином:

```
public void consume ();
```

Метод, що перевіряє, чи не використовувалася споживання події:

```
public boolean isConsumed ();
```

Метод, що повертає модифікатор для даної події (його значення збігається з однією з визначених вище змінних):

```
public int getModifiers ();
```

Метод, що отримує текстове значення для натиснутих клавіш модифікаторів, наприклад, "**Shift**", "**Button1**" або "**Ctrl** + **Shift**":

```
public static String getModifiersExText (int modifiers);
```

Метод, що дозволяє отримати розширені значення модифікаторів, які представляють стан всіх клавіш-модифікаторів і кнопок миші:

```
public int getModifiersEx ()
```

Метод, що дозволяє отримати позначку про час виникнення події (в мілісекундах з 1 січня 1970 року):

```
public long getWhen ()
```

Методи, що дозволяють визначити чи були натиснуті відповідно клавіші **Alt**, **Ctrl** або **Shift** під час виникнення події:

```
public boolean isAltDown ()
public boolean isControlDown ()
public boolean isShiftDown ()
```

3.3.2.6. Клас KeyEvent і інтерфейс KeyListener

Клас **KeyEvent** обробляє події клавіатури і містить наступні статичні **final** змінні типу **int**:

KEY_PRESSED - клавіша натиснута;

KEY_RELEASED - клавіша відпущена;

KEY_TYPED - введений символ;

KEY_LOCATION_LEFT - клавіша розташована зліва на клавіатурі (для декількох однакових клавіш);

KEY_LOCATION_RIGHT - клавіша розташована праворуч на клавіатурі (для декількох однакових клавіш);

KEY_LOCATION_NUMPAD - клавіша розташована на цифровій клавіатурі (для декількох однакових клавіш);

KEY_LOCATION_STANDARD - положення клавіші стандартне (для тих клавіш, у яких немає збігаються з ними клавіш);

KEY_LOCATION_UNKNOWN - положення клавіші не визначилося (зазвичай для події **KEY_TYPED**).

Крім того, в класі **KeyEvent** визначені коди всіх клавіш у вигляді констант, називаемих віртуальними кодами клавіш (***virtual key codes***), наприклад, **VK_F1**, **VK_SHIFT**, **VK_A**, **VK_B**, **VK_PLUS**. Вони перераховані в документації до класу **KeyEvent**. Фактичне значення віртуального коду залежить від мови та раскладкі клавіатури.

Метод, що виводить значення символу:

```
public char getKeyChar ();
```

Метод, що виводить значення коду символу:

```
public int getKeyCode ();
```

Метод, що виводить положення клавіші на клавіатурі (його можна порівняти з однією з наведених вище змінних класу):

```
getKeyLocation ();
```

Метод, що повертає рядок, що описує клавішу, наприклад, "**HOME**", "**F1**" або "**A**":

```
public static String getKeyText (int keyCode)
```

Метод, що повертає текст, який описує клавішу-модифікатор, наприклад, "**Shift**" або "**Ctrl + Shift**":

```
public static String getKeyModifiersText (int modifiers)
```

Метод, що перевіряє, чи не є клавіша, яка викликала подію, клавішею «дії» (зазвичай клавіші «дії» - це клавіші, які не є клавішами символів або клавішами модифікаторів, наприклад, клавіша **Esc**):

```
public boolean isActionKey ();
```

Метод, що дозволяє встановити значення символу:

```
public void setKeyChar (char keyChar);
```

Метод, що дозволяє встановити значення коду символу:

```
public void setKeyCode (int keyCode);
```

Метод, що дозволяє встановити значення клавіші-модифікатора:

```
public void setModifiers (int modifiers);
```

Крім зазначених властивостей і методів, клас **KeyEvent** успадковує всі властивості і методи класу **InputEvent**.

Інтерфейс **KeyListener** містить оголошення наступних методів:

• Обробка події введення символу:

```
public void keyTyped (KeyEvent e);
```

• Обробка події натискання клавіші:

```
public void keyPressed (KeyEvent e);
```

• Обробка події відпускання клавіші:

```
public void keyReleased (KeyEvent e);
```

3.3.2.7. Клас MouseEvent і інтерфейси MouseListener і MouseMotionListener

Клас **MouseEvent** обробляє події миші і містить наступні статичні **final** змінні типу **int**:

MOUSE_PRESSED - натискання кнопки миші;

MOUSE_RELEASED - відпускання кнопки миші;

MOUSE_CLICKED - клацання кнопкою миші;

MOUSE_MOVED - переміщення миші;

MOUSE_DRAGGED - переміщення миші з натиснутою кнопкою;

MOUSE_ENTERED - поява курсора миші в компоненті;

MOUSE_EXITED - вихід курсору миші з компонента;

BUTTON1, **BUTTON2** і **BUTTON3** - перша, друга і третя кнопки миші;

MOUSE_WHEEL - обертання коліщатка миші;

NOBUTTON - кнопка миші не використовулася під час події;

MOUSE_FIRST - перший номер у списку ідентифікаторів, використовуваних для подій миші;

MOUSE_LAST - останній номер у списку ідентифікаторів, використовуваних для подій миші.

Метод, що дозволяє визначити, яка кнопка була натиснута:

```
public int getButton ();
```

Метод, що дозволяє визначити кількість клацань кнопки:

```
public int getClickCount ();
```

Методи, що визначають точку екрану, в якій сталася подія миші або окремо координати ***x*** і ***y*** точки:

```
public Point getPoint ();
public int getX ();
public int getY ();
```

Метод, що дозволяє визначити, чи є подія миші подією виклику контекстного «спливаючого» меню (для Windows це натискання правої кнопки миші):

```
public boolean isPopupTrigger ();
```

Метод, що повертає рядок, який ідентифікує клавішу-модифікатор (тобто, "**Shift**"), якщо вона була натиснута одночасно з кнопкою миші:

public static String getMouseModifiersText (int modifiers);

Метод, що переводить точку координати точки на екрані, в якій сталася подія миші, в нову позицію з горизонтальним зміщенням ***x*** і вертикальним зміщенням ***y***:

```
public void translatePoint (int x, int y);
```

Крім зазначених властивостей і методів, клас **MouseEvent** успадковує всі властивості і методи класу **InputEvent**.

В інтерфейсі **MouseListener** оголошені наступні методи:

Обробка події клацання миші:

```
public void mouseClicked (MouseEvent e);
```

Обробка події натискання кнопки миші:

```
public void mousePressed (MouseEvent e);
```

Обробка події відпускання кнопки миші:

```
public void mouseReleased (MouseEvent e);
```

Обробка події входу курсора миші в компонент:

```
public void mouseEntered (MouseEvent e);
```

Обробка події виходу курсора миші з компонента:

```
public void mouseExited (MouseEvent e);
```

Інтерфейс **MouseMotionListener** містить оголошення наступних методів:

Обробка події переміщення миші, утримуючи кнопку:

```
public void mouseDragged (MouseEvent e);
```

Обробка події переміщення миші (прі не кнопці миші):

```
public void mouseMoved (MouseEvent e) ;
```

3.3.2.8. Клас MouseWheelEvent і інтерфейс MouseWheelListener

Клас **MouseWheelEvent** обробляє події, пов'язані з обертанням коліщатка миші. Дві статичні final змінні типу int:

WHEEL_UNIT_SCROLL
WHEEL_BLOCK_SCROLL

представляють прокручування коліщатка в «одиницях» (як переміщення за допомогою клавішей стрілок) і в «блоках» (як переміщення за допомогою клавіш **PageUp** і **PageDown**).

Метод, що дозволяє отримати тип обертання (що дорівнює одній з наведених вище констант):

```
public int getScrollType ();
```

Метод, що дозволяє отримати кількість «одиниць», яке необхідно прокрутити в відповідь на цю подію:

```
public int getScrollAmount ();
```

Метод, що дозволяє отримати кількість «клацань», прокручених коліщатком миші (негативне, якщо прокручування вгору і позитивне, якщо вниз):

```
public int getWheelRotation ();
```

Метод, що дозволяє отримати кількість «одиниць» прокрутки (залежить від комп'ютерної платформи):

```
public int getUnitsToScroll ();
```

Інтерфейс **MouseWheelListener** оголошує метод для обробки події обертання коліщатка миші:

```
public void mouseWheelMoved (MouseWheelEvent e);
```

3.3.3. Класи і інтерфейси обробки високорівневих подій

3.3.3.1. Інтерфейс ActionListener і клас ActionEvent

Для класів **JButton**, **JList** і **JTextField** визначено блок прослуховування **ActionListener** з єдиним методом, що викликається, коли відбувається дія для заданого об'єкта класу Button, List або TextField:

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

Клас **ActionEvent** обробляє події, пов'язані з натисканням кнопки, вибором елемента зі списку або з натисканням клавіші **Enter** в текстовому полі.

Для класу **ActionEvent** визначені наступні статичні **final** змінні типу **int**:

ACTION_PERFORMED - дія відбулася;
ALT_MASK - натиснута клавіша Alt;
SHIFT_MASK - натиснута клавіша Shift;
CTRL_MASK - натиснута клавіша Ctrl;
ACTION_FIRST - перший номер у списку ідентифікаторів, використовуваних для подій дії;
ACTION_LAST - останній номер у списку ідентифікаторів, використовуваних для подій дії.

Метод, що дозволяє отримати командний рядок, пов'язаний з даним процессом:

```
public String getActionCommand ();
```

Метод, що дозволяє отримати ключ-модифікатор для даної дії:

```
public int getModifiers ();
```

Метод, що дозволяє отримати час лічильника (в мілісекундах) для даного дії:

```
public long getWhen ();
```

Метод, що дозволяє отримати рядок параметра, що ідентифікує дану дію (зазвичай використовується для налагодження):

```
public String paramString ();
```

3.3.3.2. Інтерфейси ItemListener, ItemSelectable і клас ItemEvent

Для класів **JComboBox**, **JCheckbox**, **JRadioButton** і **JList** визначено блок прослуховування **ItemListener** з єдиним методом, що викликається, коли відбувається вибір або відміна вибору опцій в об'єктах класу **Choice**, **Checkbox** або **List**:

```
public void itemStateChanged (ItemEvent e);
```

Клас **ItemEvent** містить змінні і методи для обробки події, пов'язані з вибором елемента або скасуванням вибору елемента.

Для класу **ItemEvent** визначені наступні статичні **final** змінні типу **int**:

SELECTED - елемент обраний;

DESELECTED - вибірка елемента скасовано;

ITEM_STATE_CHANGED - стан елемента змінено;

ITEM_FIRST - перший номер у списку ідентифікаторів, використовуваних для подій вибору елемента;

ITEM_LAST - останній номер у списку ідентифікаторів, використовуваних для подій вибору елемента.

Метод, що дозволяє отримати елемент, який викликав подію:

```
public Object getItem ();
```

Метод, що дозволяє отримати зміну стану елемента (обрано або не вибрано):

```
public int getStateChange ();
```

Метод, що дозволяє отримати рядок параметра, що ідентифікує вибір елемента (зазвичай використовується для налагодження):

```
public String paramString ();
```

Метод, що повертає об'єкт інтерфейсу **ItemSelectable**:

```
public ItemSelectable getItemSelectable ();
```

Цей інтерфейс призначений для об'єктів, в яких допустимо вибір одного або більше елементів і містить оголошення методів для додавання і видалення блоків прослуховування вибору або скасування вибору елементів зі списку:

```
public void addItemListener (ItemListener l);
public void removeItemListener (ItemListener l);
```

Метод, який повинен повертати масив обраних об'єктів або ***null***, якщо не вибрано жодного об'єкта:

```
public Object [] getSelectedObjects ();
```

3.3.4. Обробка подій в Java

Щоб блок прослуховування міг фіксувати і обробляти події він повинен бути включений за допомогою методів додавання блоків прослуховування в клас (ці методи визначені в класі **Component**).

Методи включення блоків прослуховування мають наступний загальний формат:

```
public void addXXXX (XXXX l)
```

Де **XXXX** - ім'я відповідного блоку прослуховування, наприклад включення в клас блоку прослуховування для подій клавіатури реалізується за допомогою методу

```
public void addKeyListener (KeyListener l);
```

Таким чином, для обробки події в будь-якому класі необхідно:

Зробити доступним методи, визначені у відповідному інтерфейсі або інтерфейси, оголосивши клас як реалізацію (***implementation***) відповідного інтерфейсу або інтерфейсів.

Включити необхідні блоки прослуховування за допомогою відповідного методу **addXXXX**.

Описати в програмі всі методи всіх реалізованих інтерфейсів (якщо який-небудь метод не використовується в класі, він оголошується з порожнім тілом - {}). У методах інтерфейсів зазвичай використовуються методи і властивості відповідного класу обробки подій.

Скелет програми (для обробки подій клавіатури) представлений нижче:

```
class KeyTest implements KeyListener
{
...
addKeyListener (ім'я-об'єкта-KeyListener);
...
public void keyPressed (KeyEvent e)
{
// Обробка події натискання клавіші або порожньо
}
public void keyReleased (KeyEvent e)
{
// Обробка події відпускання клавіші або порожньо
}
public void keyTyped (KeyEvent e)
{
// Обробка введення символу або порожньо
}
```

Як видно з попереднього прикладу, необхідно описувати метод, навіть якщо він не використовується. Для того, щоб в класі можна було описувати тільки ті методи, які необхідно, в JDK були введені спеціальні абстрактні класи - адаптери, що містять оголошення тих же методів, що і відповідні блоки прослуховування. Імена цих класів формуються так само, як і для блоків прослуховування, тільки замість суфікса **Listener** в них використовується суфікс **Adapter**, наприклад, **KeyAdapter** або **MouseAdapter**.

Якщо створити всередині даного класу підклас, який розширює клас відповідного адаптера (наприклад, підклас **myKeyAdapter**, що розширює клас **KeyAdapter**), то всередині нього можна перевизначити тільки ті методи адаптера, які необхідно використовувати. В цьому випадку у відповідному методі **addXXXX** як параметр можна задати екземпляр створеного підкласу, наприклад,

```
addKeyListener (new myKeyAdapter ());
```

Компактнішим записом, який часто використовується програмістами на Java, є безпосереднє визначення безіменного внутрішнього класу в параметрі відповідного методу **addXXXX**, наприклад:

```
addKeyListener
(
        new KeyAdapter ()
{
// Опис методів класу KeyAdapter
}
);
```

3.4. Програмування графічних додатків в NetBeans IDE 6.1

3.4.1. Створення графічного додатку

Для створення графічного додатку необхідно відкрити новий проект за допомогою команди **New Project** меню **File** або кнопки на панелі інструментів. У вікні в полі **Categories** вибирається пункт **General**, а в полі **Projects** - пункт **Java Application**. Після натискання кнопки **Next** у вікні **New Java Application** в полі **Project Name** задається ім'я проекту, а в полі **Project Location** - шлях до папки проекту. Перемикач **Set as Main Project** потрібно включити, а перемикач **Create Main Class** - вимкнений. Після натискання кнопки **Finish** створюється новий проект.

У контекстному меню нового проекту виберіть команду **New** і в меню цієї команди виберіть команду **JFrame Form**. В полі **Class Name** вікна **New JFrame Form** задайте ім'я класу графічної форми і натисніть кнопку **Finish**.

3.4.2. Проектування графічного додатку

Після створення графічної форми в панелі інструментів вікна редактора з'являються кнопки **Source** (для перегляду і редагування вихідного тексту програми) і **Design** (створення і редагування форми в графічному

режимі). У лівій частині екрана з'являється вікно **Palette** (Палітра) з компонентами **Swing** і вікно **Properties** (Властивості) (якщо цих вікон немає на екрані, їх можна вивести за допомогою команд **Palette** і **Properties** меню Window).

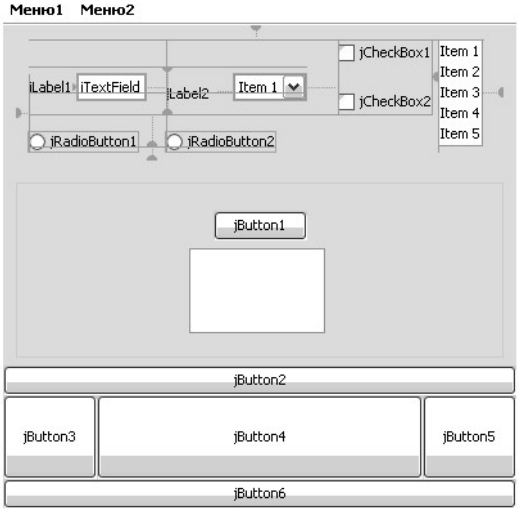
У вікні **Properties** виводяться властивості виділеного компонента **Swing** (на початку це властивості компонента **JForm**). Для компонента **JForm** можна задати властивість **title** (найменування вікна графічного додатку) та змінити властивість **background** (колір фону вікна).

Проектування графічного додатку в середовищі NetBeans IDE 6.1 включає в себе наступні етапи:

1. Перетягування (вставка) необхідного графічного компонента з вікна **Palette** в той компонент, в якому буде виводитися даний компонент, тобто батьківський компонент (спочатку батьківським компонентом є **JForm**);
2. Виконання операцій над компонентами;
3. Зміна властивостей графічного компонента;
4. Завдання (якщо необхідно) подій і їх обробки для графічного компонента;
5. Завдання (якщо необхідно) додаткового програмного коду для графічного компонента.

3.4.2.1. Вставка графічних компонент у вікно програми

Результат вставки графічних компонент у вікно програми наведено на рис. 3.4.1.



Ривсунок 3.4.1. Вікно проектування графічного додатку

При цьому у вікні Inspector (Інспектор) буде виводитися ієрархічна структура графічних компонентів програми (рис. 3.4.2).

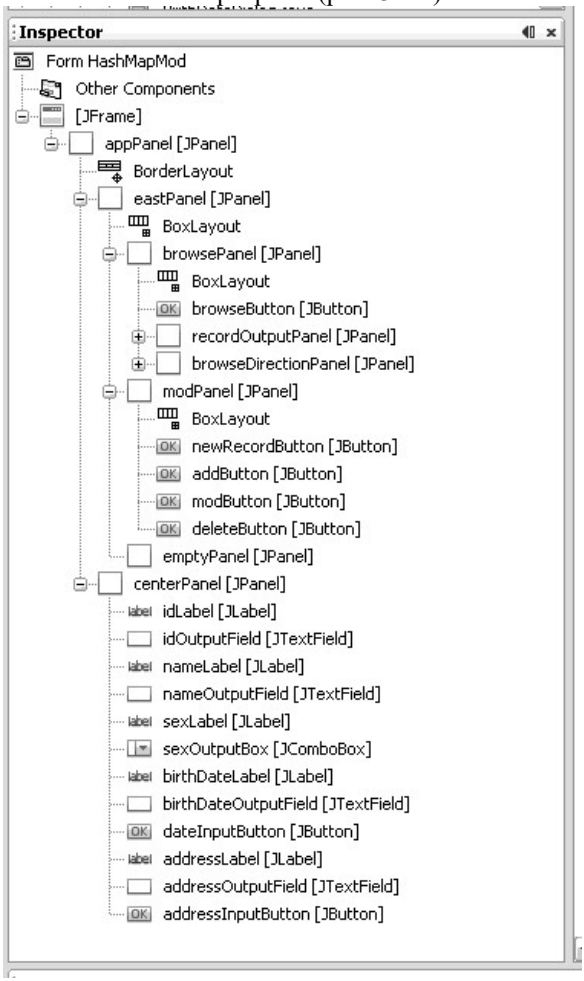


Рисунок. 3.4.2. Ієрархічна структура графічних компонент для вікна програми на рис. 3.4.1

При перетягуванні графічний компонент буде або в тому місці вікна, куди він поміщений («вільне» проектування, як для вмісту панелей **jPanel1** і **jPanel2**), або відповідно до заданого менеджером компонування (як для вмісту панелі **jPanel3**). Для вибору менеджера компоновки треба виділити компонент-контейнер (наприклад, **JForm** або **JPanel**) і в контекстному меню задати в команді **Set Layout** найменування менеджера компоновки для цього компонента.

При вставці компонентів у вікно програми їм автоматично присвоюються імена (наприклад, **JMenu1**, **JMenu2** і т.д.). Ім'я графічного компонента можна змінити, якщо викликати контекстне меню для цього компонента у вікні проектування і в діалоговому вікні команди **Change Variable Name** задати нове ім'я компонента.

Відредагувати текст для компонентів з текстом (наприклад, **JTextField** або **JButton**) можна в діалоговому вікні команди **Edit Text** контекстного меню.

Попередній перегляд вікна програми можна виконати за допомогою кнопки **Preview Design**  - на панелі інструментів проектування.

3.4.2.2. Виконання операцій над компонентами

При використанні «вільного» проектування графічні компоненти можна вирівнювати відносно один одного. Для цього ці компоненти необхідно виділити, клацаючи по ним мишею, утримуючи клавішу **Ctrl**, а потім застосувати до них одну або декілька операцій вирівнювання за допомогою команд підменю команди **Align** контекстного меню, або за допомогою кнопок на панелі інструментів.

Команда **Left to Column** або кнопка  вирівнює компоненти зліва, а команда **Right to Column** або кнопка  - праворуч по відношенню до вертикальної лінії (колонки).

Команда **Top to Row** або кнопка  вирівнює компоненти вгору, а команда **Bottom to Row** або кнопка  **-** вниз по відношенню до горизонтальної лінії (рядку).

Команда **Center horizontally** або кнопка  вирівнює компоненти по центру горизонтально а команда **Center vertically** або кнопка  **-** по центру вертикально.

Прив'язка компонент до кордонів контейнера задається за допомогою команд підменю команди **Anchor** контекстного меню.

Виділені компоненти можна зробити однакового розміру, якщо в команді **Same Size** контекстного меню включити один або обидва перемикача (**Same Width** і **Same Height**). Для повернення до розміру за замовчуванням для виділених компонент треба виконати команду **Set Default Size** контекстного меню.

Для можливості автоматичної зміни розмірів виділених компонент при виконанні додатка треба або натиснути кнопки  і / або  для змін по горизонталі і вертикалі, або включити один або обидва перемикача (**Horizontal** і **Vertical**) в команді **Auto Resizing** контекстного меню.

Вільний простір навколо компонента і можливості його зміни можна задати в діалоговому вікні команди **Space Around Component** контекстного меню.

За допомогою команд контекстного меню **Move Up** і **Move Down** контекстного меню можна перемістити компоненти вгору або вниз по ієрархічній структурі. У вікні Inspector перемістити компоненти можна, просто перетягнувши їх на нове місце.

Вирізати, копіювати, вставити або видалити виділені компоненти можна за допомогою команд **Cut**, **Copy** і **Delete** контекстного меню. Вставка вирізаних або скопійованих компонент в те місце, де знаходиться курсор миші, виконується за допомогою команди Paste (вставлений компонент буде мати ті ж властивості, що й оригінальний компонент, але при цьому компоненту буде присвоєно нове ім'я).

3.4.2.3. Зміна властивостей компонент

Після вставки компонента йому можна змінити властивості. Ця операція виконується для виділеного компонента у вікні **Properties**.

Деякі властивості можна змінити безпосередньо в рядку властивості, якщо значення властивості - число, рядок або якщо властивість задається за допомогою меню, що розкривається. Однак більшість властивостей змінюється після натискання кнопки  для властивості у відповідному діалоговому вікні.

Для кожного компонента визначено свій набір властивостей. Для виведення підказки для властивості треба навести на її найменування курсор миші, або клацнути мишею по найменуванню властивості (при цьому внизу вікна виводиться текст підказки). Найбільш часто використовувані властивості для різних груп компонент наведені в табл. 3.4.1.

Таблиця 3.4.1. Найбільш часто використовувані властивості для різних груп компонентів

Властивість	Група компонентів	Призначення
background	Практично всі компоненти.	Установка кольору фону.
font	Компоненти, що містять текст.	Установка шрифту для тексту.
text	Компоненти, що містять текст.	Завдання тексту, що виводиться в компоненті.
foreground	Компоненти, що містять текст.	Завдання кольору тексту, що виводиться в компоненті.
horizontalAlignment	Компоненти, що містять текст.	Завдання вирівнювання тексту в компоненті.
toolTipText	Всі компоненти.	Завдання тексту підказки, що виводиться при наведенні курсору миші на компонент.
border	Практично всі компоненти.	Завдання виду і параметрів рамки навколо компонента.

Властивість	Група компонентів	Призначення
buttonGroup	JCheckBox, JRadioButton, JButton	Завдання групи, до якої відноситься компонент.
selected	JCheckBox, JRadioButton	Завдання вибору для компонента.
icon	JLabel, JButton	Завдання зображення для компонента.
model	JComboBox, JList, JSpinner	Завдання списку пунктів (значень) для компонента з можливістю додавання, редагування та видалення пунктів.
selectedItem	JComboBox, JList, JSpinner	Завдання попередньо обраного елемента в списку.
editable	JComboBox, JTextField, JTextArea, JTextPane	Завдання можливості редагування вмісту компонента.
columns, rows	JTextArea	Завдання кількості стовпців і рядків в текстовій області.
selectionMode	JList	Завдання можливості множинного вибору в списку.
maximum, minimum	JSlider	Завдання максимального і мінімального значень для повзунка.
majorTickSpacing, minorTickSpacing	JSlider	Завдання кількості одиниць у великій і малій ризики шкали повзунка.
value	JSlider	Завдання початкового значення для повзунка.
dividerLocation, dividerSize, orientation	JSplitPane	Завдання розташування роздільника, його розміру (товщини) і орієнтації (горизонтальної або вертикальної).
oneTouchExpandable	JSplitPane	Завдання можливості управління виводу розщеплених панелей.
caretColor caretPosition	JTextPane	Завдання кольору і позиції курсора у вікні панелі.

Для створення групи компонентів треба перетягнути у вікно проектування компонентів **ButtonGroup** (нового компонента у вікні при

цьому не з'явиться, а з'явиться відповідний компонент у вікні **Inspector**). Для додавання будь-якого компонента в групу треба в цьому компоненті у властивості **buttonGroup** вибрати найменування групи.

При створенні обертового списку з числовими значеннями треба в діалоговому вікні властивості **model** натиснути кнопку **Advanced**, потім в діалоговому вікні включити перемикач **Generate Pre-initialization Code** ввести пропозицію створення об'єкта класу **SpinnerNumberModel**, наприклад:

```
yearModel = new SpinnerNumberModel (1990, 100, 2020, 1);
```

і натиснути кнопку **OK**. Після цього в діалоговому вікні властивості model натиснути радіокнопку **User Code** і ввести в текстовій області ім'я об'єкта (наприклад, **yearModel**).

Якщо для обертового списку задається строкова модель, то попередньо повинен бути заданий масив рядків (наприклад, **months**), а потім у вікні **Advanced** вводиться пропозицію створення об'єкта класу **SpinnerListModel**, наприклад:

```
monthModel = new SpinnerListModel (months);
```

і натискається кнопка **OK**, а потім, так само, як і в попередньому випадку, в текстовій області вводиться ім'я об'єкта (наприклад, **monthModel**).

3.4.2.3. Програмування обробки подій

Створення методу обробки події для виділеного компонента виконується за допомогою команди **Events** контекстного меню об'єкта. У підменю цієї команди спочатку вибирається подія, наприклад **Action** (клас **ActionEvent**) або **Mouse** (клас **MouseEvent**). Потім в підменю події вибирається відповідний метод класу (для класу **ActionEvent** - це єдиний метод **actionPerformed**, а для класу **MouseEvent** - один з методів: **MouseClicked**, **MouseEntered**, **MouseExited**, **MousePressed** або **MouseReleased**).

Після вибору методу відбувається перехід у вікно **Source**, де курсор миші встановлюється всередині опису методу. Програмний код обробки події задається вручну з використанням як властивостей і методів відповідного класу події, так і властивостей і методів для інших компонент.

Метод обробки події можна також створити, використовуючи вкладку **Events** вікна **Properties** для виділеного компонента. Спочатку вибирається один з методів обробки події, а потім (після натискання кнопки) в діалоговому вікні задається ім'я методу-обробника (**handler**) даної події. Таким чином, можна задати один метод-обробник для декількох компонент.

3.4.2.4. Завдання додаткового програмного коду для компонент

Використовуючи вкладку **Code** вікна **Properties** для виділеного компонента, можна задати додатковий програмний код для компонента: до створення компонента (Pre-Creation Code), після створення (**Post-Creation Code**), до ініціалізації компонента (**Pre-Init Code**) і після ініціалізації (**Post -**

Init Code), а також задати своє код для створення компонента (**Custom Creation Code**). Крім того, можна змінити найменування об'єкта (**Variable Name**) для компонента і задати свої модифікатори для об'єкта (**Variable Modifiers**).

4. Порядок виконання роботи

Створити графічний додаток з використанням засобів проектування пакета NetBeans IDE 6.1.

Варіант 1

Введення тексту в графічне вікно програми. У вікні визначена рядок меню (JMenuBar), в якій визначено два меню (JMenu) - "Шрифт" і "Стиль". В меню "Шрифт" визначено три пункти меню (JRadioButtonMenuItem): "Times New Roman" (шрифт за замовчуванням), "Arial" і "Verdana". В меню "Стиль" визначено чотири пункти меню (JRadioButtonMenuItem): "Простий" (шрифт за замовчуванням), "Жирний", "Курсив" і "Жирний курсив". У текстовій панелі (JTextPane) "Введення тексту" вікна програми вводиться текст, що набирається на клавіатурі.

При виборі одного з пунктів меню текст в панелі виводиться відповідним шрифтом і / або стилем.

Варіант 2

Введення тексту в графічне вікно програми. У верхній панелі (JPanel) вікна "Характеристики шрифту" визначено такі компоненти: напис (JLabel) "Шрифт:", спливаюче меню (JComboBox), в якому три пункти: "Times New Roman" (шрифт за замовчуванням), "Arial" і "Verdana", напис (JLabel) "Стиль:" і розкривається меню (JComboBox), в якому чотири пункти: "Простий" (шрифт за замовчуванням), "Жирний", "курсив" і "Жирний курсив". У нижній текстової панелі (JTextPane) "Введення тексту" вікна програми вводиться текст, що набирається на клавіатурі.

При виборі одного з пунктів меню текст в панелі виводиться відповідним шрифтом і / або відповідного стилю.

Варіант 3

Зміна розміру фігури в графічному вікні. У вікні визначена рядок меню (JMenuBar), в якій визначено меню (JMenu) "Параметр" і "Зміна". В меню "Параметр" визначено два пункти меню (JCheckBoxMenuItem): "Ширина" (за замовчуванням включена) і "Висота" (за замовчуванням включена). В меню "Зміна" визначено два пункти меню (JMenuItem): "Збільшити" і "Зменшити". У центрі панелі (JPanel) вікна "Виведення зображення" в графічному контексті задається (за допомогою методу drawRect ()) прямокутник.

При виборі одного з пунктів другого меню фігура стрибкоподібно (на 10 пікселів) збільшується або зменшується для заданого параметра або параметрів в першому меню і перемальовується. Якщо в першому меню не включено жодного з пунктів, фігура не змінюється.

Варіант 4

Зміна розміру фігури в графічному вікні. У центрі верхньої панелі (JPanel) "Вивід зображення" в графічному контексті задається (за допомогою методу drawRect ()) прямокутник. У нижній панелі (JPanel) "Зміна розміру" задається напис (JLabel) "Параметр:", два перемикача (JCheckBox): "Ширина:" (обидва включені за замовчуванням) і "Висота:", а також дві кнопки (JButton): "Збільшити" і "Зменшити".

При виборі однієї з радіокнопок і натисканні однієї з двох кнопок фігура стрибкоподібно (на 10 пікселів) збільшується або зменшується для заданого параметра і перемальовується. Якщо обидва перемикача вимкнені, фігура не змінюється.

Варіант 5

Виведення рядка заданим шрифтом і заданого кольору в графічному вікні. У верхній панелі (JPanel) "Управління виведенням" задається напис (JLabel) "Текст:" і текстове поле (JTextField), напис (JLabel) "Гарнітура:", що обертається список (JSpinner) зі значеннями "Times New Roman" (шрифт по замовчуванням), "Arial" і "Verdana", напис (JLabel) "колір:" і обертається список (JSpinner) зі значеннями "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і "Синій", а також кнопка (JButton) "Вивести рядок". У нижній панелі (JPanel) "Виведення рядку" виводиться в графічному контексті (за допомогою методу drawString ()) в довільному місці порожній рядок.

При завданні тексту рядка в текстовому полі, параметрів рядка в обертових списках верхній панелі і натисканні кнопки "Виведення рядка" рядок заданого вмісту і кольору виводиться заданим шрифтом в нижній панелі.

Варіант 6

Виведення рядка заданим шрифтом і заданого кольору в графічному вікні. У верхній панелі (JPanel) "Управління виведенням" задається напис (JLabel) "Рядок:" і текстове поле (JTextField), напис (JLabel) "Гарнітура:", спливаюче меню (JComboBox), в якому три пункти: "Times New Roman "(шрифт за замовчуванням)," Arial "і" Verdana ", напис (JLabel)" колір: "і розкривається меню (JComboBox), в якому чотири пункти:" Чорний "(колір за замовчуванням)," Червоний "," Зелений "і" Синій ", а також кнопка (JButton)" Вивести рядок ". У нижній панелі (JPanel) "Виведення рядку" виводиться в графічному контексті (за допомогою методу drawString ()) в довільному місці порожній рядок.

При завданні тексту рядка в текстовому полі, параметрів рядка в розкривних меню верхньої панелі і натисканні кнопки "Виведення рядка" рядок заданого вмісту і кольору виводиться заданим шрифтом в нижній панелі.

Варіант 7

Виведення фігур в графічному вікні. У вікні визначена рядок меню (JMenuBar), в якій визначено два меню (JMenu) - "Колір" і "Виведення". У меню" Колір "визначено чотири пункти меню (JRadioButtonMenuItem):" Чорний "(колір за замовчуванням)," Червоний " , "Зелений" і "Синій". у меню "Виведення" визначено три пункти меню (JCheckBoxMenuItem): "Квадрат", "Коло", і "Трикутник". в панелі (JPanel) вікна виводяться в графічному контексті (за допомогою методів drawRect (), drawOval () і drawPolygon ()) три фігури: прямокутник, коло і трикутник.

При виборі пункту першого меню всі фігури заповнюються заданим кольором. Команди другого меню відключають або включають виведення відповідної фігури на екран.

Варіант 8

Виведення фігур в графічному вікні. У вікні визначена рядок меню (JMenuBar), в якій визначено два меню (JMenu) - "Колір" і"Виведення". У меню" Колір "визначено чотири пункти меню (JRadioButtonMenuItem):" Чорний "(колір за замовчуванням)," Червоний " , "Зелений" і "Синій". у меню "Виведення" визначено три пункти меню (JCheckBoxMenuItem): "Квадрат", "Коло", і "Трикутник". в вірніше панелі вікна панелі (JPanel) вікна виводяться в графічному контексті (з допомогою методів drawRect (), drawOval () і drawPolygon ()) три фігури: прямокутник, коло і трикутник. В нижній панелі (JPanel) "Управління виведенням" задається напис (JLabel) "Колір:", спливаюче меню (JComboBox), В якому визначені чотири пункти: "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і "Синій", напис (JLabel) "Виведення:" і три перемикача (JCheckBox): "Квадрат", "Коло " і" Трикутник ".

При виборі пункту меню, що розкривається все фігури заповнюються заданим кольором. На екран виводяться тільки ті фігури, перемикачі яких включені.

Варіант 9

Зміна розміру зображення в графічному вікні. У верхній панелі вікна (JPanel) вікна "Виведення зображення" в компоненті (JLabel) задається довільне зображення. У нижній панелі (JPanel) "Управління виведенням" задаються дві кнопки (JButton): "Збільшити" і "Зменшити".

При натисканні першої кнопки зображення збільшується на 10 пікселів по ширині і висоті, при натисканні другої кнопки - зменшується на ту ж кількість пікселів.

Варіант 10

Зміна розміру зображення в графічному вікні. В панелі вікна (JPanel) вікна "Виведення зображення" в компоненті (JLabel) задається довільне зображення. В панелі інструментів вікна задаються дві кнопки (JButton): "Збільшити" і "Зменшити".

При натисканні першої кнопки зображення збільшується на 10 пікселів по ширині і висоті, при натисканні другої кнопки - зменшується на ту ж кількість пікселів.

Варіант 11

Переміщення фігури в графічному вікні. У центрі верхньої панелі (JPanel) "Виведення зображення" в графічному контексті задається (за допомогою методу drawRect ()) прямокутник. У нижній панелі (JPanel) "Переміщення зображення" задаються наступні компоненти: напис (JLabel) "Колір:", спливаюче меню (JComboBox), в якому визначені чотири пункти: "Чорний" (колір за замовчуванням), "Червоний", "Зелений "і" Синій ", а також чотири кнопки (JButton):" Ліворуч "," Вгору "" Вправо "і" вниз ".

При виборі кольору прямокутник перемальовується заданим кольором, а при натисканні однієї з чотирьох кнопок зображення стрибкоподібно (на 10 пікселів) переміщається в цьому напрямку і перемальовується.

Варіант 12

Переміщення фігури в графічному вікні. У центрі верхньої панелі (JPanel) "Виведення зображення" в графічному контексті задається (за допомогою методу drawRect ()) прямокутник. У нижній панелі (JPanel) "Переміщення зображення" задаються наступні компоненти: напис (JLabel) "Колір:", що обертається список (JSpinner) зі значеннями "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і "Синій" , а також чотири перемикача (JRadioButton): "Ліворуч", "Вгору" "Вправо" і "вниз" і кнопка (JButton) "Перемістити".

При виборі кольору прямокутник перемальовується заданим кольором, а при натисканні однієї з чотирьох радіокнопок зображення стрибкоподібно (на 10 пікселів) переміщається в цьому напрямку і перемальовується.

Варіант 13

Введення тексту в графічне вікно програми. У верхній панелі (JPanel) вікна "Характеристики шрифту" визначено такі компоненти: напис (JLabel) "Колір:", спливаюче меню (JComboBox), в якому чотири пункти: "Чорний" (колір за замовчуванням), "Червоний", "Зелений "і" Синій ", напис (JLabel)" Розмір: "і три радиокнопки (JRadioButton) з написами:" 12pt "(розмір за замовчуванням)," 14pt "і" 16pt ". У нижній текстової панелі (JTextPane)

"Введення тексту" вікна програми вводиться текст, що набирається на клавіатурі.

При виборі одного з пунктів меню текст в панелі виводиться відповідним кольором і / або відповідного розміру.

Варіант 14

Введення тексту в графічне вікно програми. В панелі інструментів вікна (JToolBar) визначені наступні компоненти (JButton) з написами: "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і "Синій", "12pt" (розмір за замовчуванням), "14pt" і "16pt". У текстовій панелі (JTextPane) "Введення тексту" вікна програми вводиться текст, що набирається на клавіатурі. Написи в кнопках повинні бути виконані відповідним кольором і шрифтом.

При виборі на панелі інструментів кольору і / або розміру текст в панелі виводиться відповідним кольором і / або відповідного розміру.

Варіант 15

Зміна фігури в графічному вікні. У центрі верхньої панелі (JPanel) "Виведення прямокутника" в графічному контексті малюється (за допомогою методу drawRoundRect ()) прямокутник з округленими вершинами. У нижній панелі (JPanel) "Параметри прямокутника" задаються наступні компоненти: напис (JLabel) "Колір:", що обертається список (JSpinner) зі значеннями: "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і "Синій ", напис (JLabel)" Ширина заокруглення ", текстове поле (JTextField), напис (JLabel)" Висота заокруглення: ", текстове поле (JTextField) і кнопка (JButton)" Змінити прямокутник ".

При наборі даних в обертових списках, введенні даних в текстових полях (в пікселях) і при натисканні кнопки "Змінити прямокутник" прямокутник перемальовується заданим кольором і з заданим новими значеннями ширини і висоти заокруглень. Перед виведенням фігури виконується перевірка, чи введені в текстових полях всі дані і чи є вони цілими числами.

Варіант 16

Зміна фігури в графічному вікні. У центрі верхньої панелі (JPanel) "Виведення прямокутника" в графічному контексті малюється (за допомогою методу drawRoundRect ()) прямокутник з округленими вершинами. У нижній панелі (JPanel) "Параметри прямокутника" задаються наступні компоненти: напис (JLabel) "Колір:", що розкриває меню (JComboBox) зі значеннями: "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і "Синій ", напис (JLabel)" Ширина заокруглення ", що обертається список (JSpinner), напис (JLabel)" Висота заокруглення: ", що обертається список (JSpinner) і кнопка (JButton)" Змінити прямокутник ".

При наборі даних в обертових списках, введенні даних в текстових полях (в пікселях) і при натисканні кнопки "Змінити прямокутник" прямокутник перемальовується заданим кольором і з заданим новими значеннями ширини і висоти заокруглень. Діапазон зміни обертових списків - від 0 до 30 з кроком 1.

Варіант 17

Малювання фігури в графічному вікні. У верхній панелі (JPanel) "Виведення дуги" в графічному контексті малюється (за допомогою методу drawArc ()) дуга. У нижній панелі (JPanel) "Параметри дуги" задаються наступні компоненти: напис (JLabel) "Координата X:", текстове поле (JTextField), напис (JLabel) "Координата Y:", текстове поле (JTextField), напис (JLabel) "Ширина:", текстове поле (JTextField), напис (JLabel) "Висота:", текстове поле (JTextField), напис (JLabel) "Початковий кут:", текстове поле (JTextField), напис (JLabel) "Кінцевий кут: ", текстове поле (JTextField) і кнопка (JButton)" Вивести дугу ".

При введенні даних (в пікселях і в градусах - для кутів) і при натисканні кнопки "Вивести дугу" дуга перемальовується з заданими параметрами. При введенні даних перевіряється, чи всі дані введені і чи є дані цілими числами. Значення кутів повинні бути позитивними або негативними цілими числами в діапазоні (по абсолютній величині) від 0 до 360.

Варіант 18

Малювання фігури в графічному вікні. У верхній панелі (JPanel) "Виведення дуги" в графічному контексті малюється (за допомогою методу drawArc ()) дуга. У нижній панелі (JPanel) "Параметри дуги" задаються наступні компоненти: напис (JLabel) "Координата X:", текстове поле (JTextField), напис (JLabel) "Координата Y:", текстове поле (JTextField), напис (JLabel) "Ширина:", текстове поле (JTextField), напис (JLabel) "Висота:", текстове поле (JTextField), напис (JLabel) "Початковий кут:", дві радіокнопки "Плюс" і "Мінус", що обертається список (JSpinner), напис (JLabel) "Кінцевий кут:", дві радіокнопки "Плюс" і "Мінус", що обертається список (JSpinner) і кнопка (JButton) "Вивести дугу".

При введенні даних для координат і розміру (в пікселях), набору даних для кутів і при натисканні кнопки "Вивести дугу" дуга перемальовується з заданими параметрами. При введенні даних перевіряється, чи всі дані введені і чи є дані цілими числами. Значення даних в обертовому списку - в діапазоні від 0 до 360 з кроком 45.

Варіант 19

Зміна фігури в графічному вікні. У центрі верхньої панелі (JPanel) "Виведення трикутника" в графічному контексті малюється (за допомогою методу drawPolygon ()) трикутник. У нижній панелі (JPanel) "Параметри

трикутника" задаються наступні компоненти: напис (JLabel) "Колір:", спливаюче меню (JComboBox) з пунктами: "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і "Синій ", напис (JLabel)" Номер вершини: ", спливаюче меню (JComboBox) з пунктами" 1 "," 2 "і" 3 ", напис (JLabel)" Координата X: ", що обертається список (JSpinner), напис (JLabel) "Координата Y:", що обертається список (JSpinner) і кнопка (JButton) "Змінити трикутник".

При наборі даних в обертових списках (в пікселях) і при натисканні кнопки "Змінити трикутник" трикутник перемальовується заданим кольором і з заданим новим положенням однієї з вершин. Діапазон зміни обертових списків - від 0 до 150 з кроком 1.

Варіант 20

Зміна фігури в графічному вікні. У центрі верхньої панелі (JPanel) "Виведення трикутника" в графічному контексті малюється (за допомогою методу drawPolygon ()) трикутник. У нижній панелі (JPanel) "Параметри трикутника" задаються наступні компоненти: напис (JLabel) "Колір:", що обертається список (JSpinner) зі значеннями: "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і "Синій ", напис (JLabel)" Номер вершини: ", що обертається список (JSpinner) зі значеннями" 1 "," 2 "і" 3 ", напис (JLabel)" Координата X: ", текстове поле (JTextField), напис (JLabel) "Координата Y:", текстове поле (JTextField) і кнопка (JButton) "Змінити трикутник".

При наборі даних в обертових списках (в пікселях), введення даних в текстових полях (в пікселях) і при натисканні кнопки "Змінити трикутник" трикутник перемальовується заданим кольором і з заданим новим положенням однієї з вершин. Перед виведенням фігури виконується перевірка, чи введені в текстових полях всі дані і чи є вони цілими числами.

Варіант 21

Зміна написи в графічному вікні. У центрі верхньої панелі (JPanel) "Виведення напису" в графічному контексті виводиться (за допомогою методу drawString ()) довільна напис. У нижній панелі (JPanel) "Параметри напису" задаються наступні компоненти: напис (JLabel) "Текст:", текстове поле (JTextField), напис (JLabel) "Колір:", що обертається список (JSpinner) зі значеннями: "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і "Синій", напис (JLabel) "Розмір:", що обертається список (JSpinner) зі значеннями "10pt", "12pt" (розмір за замовчуванням), "14pt" і "16pt", напис (JLabel) "Координата X:", текстове поле (JTextField), напис (JLabel) "Координата Y:", текстове поле (JTextField), а також кнопка (JButton) "Вивести рядок".

Спочатку в текстових полях "Координата X:" і "Координата Y:" встановлюються координати початку базової лінії напису (текстові поля задані як Нередагована). При введенні даних напис змінює текст, колір і / або

розмір. При натисканні мишею в області верхньої панелі в текстових полях виводяться координати точки клацання, і напис перемальовується в цій точці.

Варіант 22

Зміна написи в графічному вікні. У центрі верхньої панелі (JPanel) "Виведення напису" в графічному контексті виводиться (за допомогою методу drawString ()) довільна напис. У нижній панелі (JPanel) "Параметри напису" задаються наступні компоненти: напис (JLabel) "Текст:", текстове поле (JTextField), напис (JLabel) "Колір:", чотири радиокнопки (JRadioButton) зі значеннями: "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і "Синій", що обертається список (JLabel) "Розмір:", три радиокнопки (JRadioButton) зі значеннями: "10pt", "12pt" (розмір за замовчуванням) і "14pt ", напис (JLabel)" Координата X: ", текстове поле (JTextField), напис (JLabel)" Координата Y: ", текстове поле (JTextField), а також кнопка (JButton)" Вивести рядок ".

Спочатку в текстових полях "Координата X:" і "Координата Y:" встановлюються координати початку базової лінії напису (текстові поля задані як Нередагована). При введенні даних напис змінює текст, колір і / або розмір. При натисканні мишею в області верхньої панелі в текстових полях виводяться координати точки клацання і напис перемальовується в цій точці.

Варіант 23

Малювання фігури в графічному вікні. У центрі верхньої панелі (JPanel) "Виведення еліпса" в графічному контексті малюється (за допомогою методу drawOval ()) еліпс. У нижній панелі (JPanel) "Параметри еліпса" задаються наступні компоненти: напис (JLabel) "Колір:", чотири радиокнопки (JRadioButton) з написами: "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і "Синій ", напис (JLabel)" Координата X центру: ", текстове поле (JTextField), напис (JLabel)" Координата Y центру: ", текстове поле (JTextField), напис (JLabel)" Піввісь a: ", текстове поле (JTextField), напис (JLabel) "Піввісь b:", текстове поле (JTextField) і кнопка (JButton) "Вивести еліпс".

При введенні даних в текстових полях (в пікселях) і при натисканні кнопки "Вивести еліпс" у верхній панелі малюється еліпс. Перед виведенням фігури виконується перевірка, чи введені всі дані і чи є вони числами. Для виклику методу drawOval () виконуються наступні перетворення: координата X лівого верхнього кута обчислюється як координата X центру мінус значення a, координата Y лівого верхнього кута обчислюється як координата Y центру мінус значення b, ширина - як подвоєне значення a, а висота - як подвоєне значення b.

Варіант 24

Малювання фігури в графічному вікні. У центрі верхньої панелі (JPanel) "Виведення еліпса" в графічному контексті малюється (за допомогою

методу drawOval ()) еліпс. У нижній панелі (JPanel) "Параметри еліпса" задаються наступні компоненти: напис (JLabel) "Колір:", спливаюче меню (JComboBox) з пунктами: "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і "Синій ", напис (JLabel)" Координата X центру: ", що обертається список (JSpinner), напис (JLabel)" Координата Y центру: ", що обертається список (JSpinner), напис (JLabel)" Піввісь a: ", що обертається список (JSpinner), напис (JLabel) "Піввісь b:", що обертається список (JSpinner) і кнопка (JButton) "Вивести еліпс".

При наборі даних в обертових списках (в пікселях) і при натисканні кнопки "Вивести еліпс" у верхній панелі малюється еліпс. Діапазон зміни обертових списків - від 10 до 150 з кроком 1. Для виклику методу drawOval () виконуються наступні перетворення: координата X лівого верхнього кута обчислюється як координата X центру мінус значення a, координата Y лівого верхнього кута обчислюється як координата Y центру мінус значення b , ширина - як подвоєне значення a, а висота - як подвоєне значення b.

Варіант 25

Введення тексту в графічне вікно програми. У верхній панелі вікна (JPanel) "Параметри шрифту" запропоновано такі компоненти: напис (JLabel) "Колір:", чотири радиокнопки (JRadioButton) з написами: "Чорний" (колір за замовчуванням), "Червоний", "Зелений" і " синій ", напис (JLabel)" Розмір: "і текстове поле (JTextField) (початкове значення цього текстового поля дорівнює 12). У текстовій панелі (JTextPane) "Введення тексту" вікна програми вводиться текст, що набирається на клавіатурі. Написи для радіокнопок повинні бути виконані відповідним кольором.

При виборі однієї з радіокнопок кольору і введення в текстовому полі цілого числа - розміру тексту вміст текстової панелі виводиться відповідним кольором і / або відповідного розміру. При введенні значення в текстове поле перевіряється, чи є це значення цілим числом, що містить одну або дві цифри.

Варіант 26

Введення тексту в графічне вікно програми. У верхній панелі вікна (JPanel) "Параметри шрифту" запропоновано такі компоненти: напис (JLabel) "Стиль:", спливаюче меню (JComboBox) з пунктами: "Простий" (стиль за замовчуванням), "Жирний" і "Курсив", напис (JLabel) "Розмір:" і розкривається меню (JComboBox) з пунктами: "10pt" (стиль за замовчуванням), "12pt" (розмір за замовчуванням) і "14pt". У текстовій панелі (JTextPane) "Введення тексту" вікна програми вводиться текст, що набирається на клавіатурі.

При виборі одного з пунктів меню, що розкривається стилю і введення в текстовому полі цілого числа - розміру тексту вміст текстової панелі виводиться відповідним кольором і / або відповідного розміру. При введенні

значення в текстове поле перевіряється, чи є це значення цілим числом, що містить одну або дві цифри.

Варіант 27

Малювання фігури в графічному вікні. У верхній панелі вікна (JPanel) "Координати вершин трикутника" запропоновано такі компоненти: напис (JLabel) "X1:", текстове поле (JTextField), напис (JLabel) "Y1:", текстове поле (JTextField), напис (JLabel) "X2:", текстове поле (JTextField), напис (JLabel) "Y2:", текстове поле (JTextField), напис (JLabel) "X3:", текстове поле (JTextField), напис (JLabel) "Y3:", текстове поле (JTextField) і кнопка (JButton) "Вивести трикутник". У нижній панелі вікна (JPanel) "Виведення трикутника" в графічному контексті малюється (за допомогою методу fillPolygon ()) трикутник, заповнений чорним кольором.

Малювання трикутника виконується після введення координат вершин при натисканні кнопки "Вивести трикутник". Перед виведенням трикутника перевіряється завдання всіх координат, які повинні бути позитивними цілими числами.

Варіант 28

Малювання фігури в графічному вікні. У верхній панелі вікна (JPanel) "Координати вершин трикутника" запропоновано такі компоненти: напис (JLabel) "X1:", що обертається список (JSpinner), напис (JLabel) "Y1:", що обертається список (JSpinner), напис (JLabel) "X2:", що обертається список (JSpinner), напис (JLabel) "Y2:", що обертається список (JSpinner), напис (JLabel) "X3:", що обертається список (JSpinner), напис (JLabel) "Y3:", обертається список (JSpinner) і кнопка (JButton) "Вивести трикутник". У нижній панелі вікна (JPanel) "Виведення трикутника" в графічному контексті малюється (за допомогою методу fillPolygon ()) трикутник, заповнений чорним кольором.

Малювання трикутника виконується після введення координат вершин при натисканні кнопки "Вивести трикутник". Координати вершин повинні бути в діапазоні від 0 до 100.

Варіант 29

Зміна розміру зображення в графічному вікні. У верхній панелі вікна (JPanel) "Розмір зображення" запропоновано такі компоненти: напис (JLabel) "Ширина:", текстове поле (JTextField), напис (JLabel) "Висота:", текстове поле (JTextField) і кнопка (JButton): "Вивести зображення". У нижній панелі вікна (JPanel) вікна "Виведення зображення" в компоненті (JLabel) задається довільне зображення.

При введенні розміру зображення по ширині і висоті і натисканні кнопки "Вивести зображення" в нижній панелі виводиться масштабувати зображення до заданих розмірів (в пікселях). Спочатку зображення має

«природний» розмір. Перед виведенням зображення виконується перевірка, задані чи значення ширини і висоти і чи є вони цілими числами.

Варіант 30

Зміна розміру зображення в графічному вікні. У верхній панелі вікна (JPanel) "Розмір зображення" запропоновано такі компоненти: напис (JLabel) "Ширина:", бігунок (JSlider), напис (JLabel) "Висота:", бігунок (JSlider) і кнопка (JButton): "Вивести зображення ". У нижній панелі вікна (JPanel) вікна "Виведення зображення" в компоненті (JLabel) задається довільне зображення.

При установці бігунків розміру зображення по ширині і висоті і натисканні кнопки "Вивести зображення" в нижній панелі виводиться масштабувати зображення до заданих розмірів (в пікселях). Спочатку зображення має «природний» розмір. Нижні і верхні межі бігунків повинні мати значення 10 і 150.

5. Зміст звіту

У звіті повинен бути представлений текст програми (без згенерованого коду), а також один або кілька скріншотів, що відображають виконання програми.

6. Приклад виконання завдання 4

Завдання кольору фону і кольору тексту для текстового поля (проект JavaGUIApplication1). У верхній частині вікна програми задаються дві панелі (JPanel): "Вибір кольору фону" і "Вибір кольору тексту". В панелі "Вибір кольору фону" розміщується спливаюче меню (JComboBox), в якому задаються найменування квітів фону. В панелі "Вибір кольору тексту" розміщуються радиокнопки (JRadioButton), в яких задаються найменування квітів тексту. У нижній панелі "Виведення тексту" задається текстове поле (JTextField).

При виборі кольору фону у спадному меню цей колір встановлюється як колір фону для тексту. При виборі кольору тексту за допомогою однієї з радіокнопок цей колір встановлюється як колір переднього плану текстової області.

Текст програми:

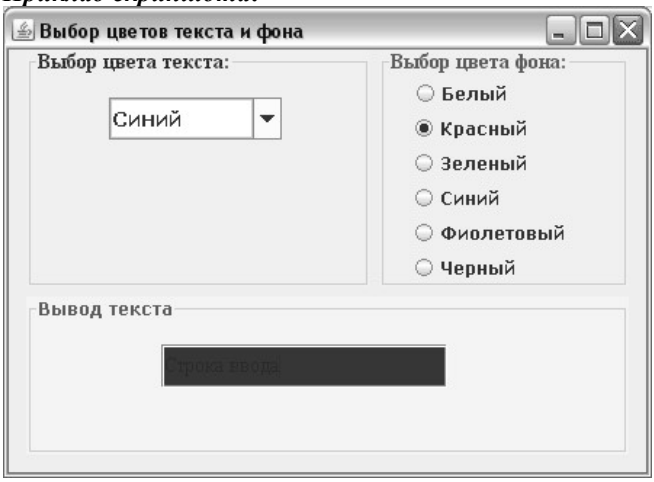
```
/ *
 * JFrameApp1.java
 * /
```

```
/ **
 *
 * @Author Шонін В.А.
 * /
import java.awt. *;
import java.awt.event. *;
import javax.swing. *;
public class JFrameApp1 extends javax.swing.JFrame {
// Обраний колір тексту
Color selectedTextColor = Color.BLACK;
// Обраний колір фону
Color selectedBackgroundColor = Color.WHITE;
// Масив найменувань кольору тексту в спадному меню
String [] textColorName = { "Чорний", "Білий", "Червоний",
"Зелений", "Синій", "Фіолетовий"};
Color [] textColor = {Color.BLACK, Color.WHITE, Color.RED,
Color.GREEN, Color.BLUE, Color.MAGENTA};
public JFrameApp1 () {
initComponents ();
}

// Обробка події вибору пункту меню, що розкривається
private void textColorChoiceItemStateChanged
java.awt.event.ItemEvent evt) {
// Якщо подія - вибір елемента списку
if (evt.getStateChange () == ItemEvent.SELECTED) {
// Отримання обраного елемента
String selectedItemValue = (String) evt.getItem ();
// Визначення обраного кольору тексту
for (int i = 0; i <textColorName.length; i ++) {
// Якщо колір знайдений
if (selectedItemValue.equals (textColorName [i])) {
// Установка нового індексу кольору фігури
textOutputField.setForeground (textColor [i]);
// Вихід з циклу
break;
}
}
}
}
// Обробка події вибору радиокнопки "Чорний"
 private void blackColorChoiceItemStateChanged
(java.awt.event.ItemEvent evt) {
// Якщо відбулося включення радиокнопки
if (evt.getStateChange () == ItemEvent.SELECTED)
// Зміна кольору малювання лінії на чорний колір
textOutputField.setBackground (Color.BLACK);
}
```

```
// Обробка події вибору радиокнопки "Фіолетовий"
private void magentaColorChoiceItemStateChanged
(java.awt.event.ItemEvent evt) {
// Якщо відбулося включення радиокнопки
if (evt.getStateChange () == ItemEvent.SELECTED)
// Зміна кольору малювання лінії на фіолетовий колір
textOutputField.setBackground (Color.MAGENTA);
}
// Обробка події вибору радиокнопки "Синій"
private void blueColorChoiceItemStateChanged
(java.awt.event.ItemEvent evt) {
// Якщо відбулося включення радиокнопки
if (evt.getStateChange () == ItemEvent.SELECTED)
// Зміна кольору малювання лінії на синій колір
textOutputField.setBackground (Color.BLUE);
}
// Обробка події вибору радиокнопки "Зелений"
private void greenColorChoiceItemStateChanged
(java.awt.event.ItemEvent evt) {
// Якщо відбулося включення радиокнопки
if (evt.getStateChange () == ItemEvent.SELECTED)
// Зміна кольору малювання лінії на зелений колір
textOutputField.setBackground (Color.GREEN);
}
// Обробка події вибору радиокнопки "Червоний"
private void redColorChoiceItemStateChanged
(java.awt.event.ItemEvent evt) {
// Якщо відбулося включення радиокнопки
if (evt.getStateChange () == ItemEvent.SELECTED)
// Зміна кольору малювання лінії на червоний колір
textOutputField.setBackground (Color.RED);
}
// Обробка події вибору радиокнопки "Білий"
private void whiteColorChoiceItemStateChanged
(java.awt.event.ItemEvent evt) {
// Якщо відбулося включення радиокнопки
if (evt.getStateChange () == ItemEvent.SELECTED)
// Зміна кольору малювання лінії на білий колір
textOutputField.setBackground (Color.BLACK);
}

// Обробка події натискання кнопки "Копіювання тексту"
public static void main (String args []) {
java.awt.EventQueue.invokeLater (new Runnable () {
public void run () {
new JFrameApp1 (). setVisible (true);
}
});
}
```

```
// Variables declaration - do not modify
private javax.swing.JPanel backgroundColorChoice;
private javax.swing.ButtonGroup backgroundColorGroup;
private javax.swing.JRadioButton blackColorChoice;
private javax.swing.JRadioButton blueColorChoice;
private javax.swing.JRadioButton greenColorChoice;
private javax.swing.JRadioButton magentaColorChoice;
private javax.swing.JRadioButton redColorChoice;
private javax.swing.JComboBox textColorChoice;
private javax.swing.JPanel textColorChoicePanel;
private javax.swing.JTextField textOutputField;
private javax.swing.JPanel textOutputPanel;
private javax.swing.JRadioButton whiteColorChoice;
// End of variables declaration
}
}
```

Приклад скриншота:



6. Зміст звіту

Текст і вивід програми, а також рядок аргументів вікна Project Properties з аргументами виклику для свого варіанту.

7. Питання для самоконтролю

1. На які типи можна розділити компоненти Swing? Коротко охарактеризуйте кожен тип.
2. Які частини містить схема MVC і як вони взаємодіють між собою? Коротко охарактеризуйте кожну частину.
3. Які інтерфейси реалізують схему MVC в бібліотеці Swing?
4. Як реалізується схема MVC в бібліотеці Swing?
5. Які компоненти кореневої панелі визначені в Swing? Як можна отримати або встановити кореневу панель?
6. Як реалізована шарувата панель в Swing (створення, змінні і методи)?
7. Які компоненти можна ставити в панелі вмісту? Як встановити і отримати панель вмісту? Як додаються компоненти в панель вмісту?
8. Для яких цілей використовується і як реалізується скляна панель в Swing?
9. Як створюється і закривається фрейм в Swing?
10. Якими способами можна створювати діалогові вікна в Swing? Коротко охарактеризуйте кожен спосіб.
11. Як створюються діалогові вікна за допомогою класу JDialog?
12. Які стандартні діалогові вікна визначені в класі JOptionPane в Swing? Як виконується створення цих діалогових вікон?
13. Як створюється внутрішній фрейм і робочий стіл в Swing? Як проводиться обробка подій у внутрішньому фреймі?
14. Як в Swing задається клас JPanel і як виконується робота з цим класом в програмах на мові Java?
15. Як в Swing реалізується виведення компонента на дисплей за допомогою класів JScrollPane і JViewport?
16. Як в Swing реалізована панель з вкладками (tabbed pane)?
17. Які кроки необхідно виконати в програмі для реалізації панелі з вкладками?
18. Як в Swing реалізована розщеплена панель (split pane)?
19. Як в Swing реалізована панель інструментів (tool bar)?
20. Як в Swing реалізовані значки (icons) і значки із зображенням (image icons)?
21. Як в Swing реалізовані написи (labels)?
22. Які операції над кнопками можна виконувати за допомогою методів класу AbstractButton?
23. Як в Swing реалізовані кнопки (buttons)?
24. Як в Swing реалізовані прапорці (check boxes)?
25. Як в Swing реалізовані перемикачі (radiobuttons)?
26. Які види списків визначені в Swing? Коротко охарактеризуйте кожен вид списків.
27. Як в Swing реалізовано комбіноване поле (combo box)?

28. Як в Swing реалізований список (list)?
29. Як в Swing реалізований список що «обертається» (spinner)?
30. Як виконується виведення і обробка подій для списків в класі Spinner?
31. Як виконується виведення і обробка подій для чисел в класі Spinner?
32. Як виконується виведення і обробка подій для дат в класі Spinner?
33. Які можливості по обробці тексту надає клас JTextComponent?
34. Для яких цілей використовується, і які типи текстового вмісту дозволяє обробляти клас JTextPane?
35. Як задається «спливаюче» меню в Swing?
36. Як реалізується індикатор виконання (progress bar) в Swing?
37. Як реалізується смуга прокрутки (scroll bar) і бігунки (sliders) в Swing?
38. Як в Swing реалізуються «спливаючі» підказки?
39. Які компоненти меню визначені в мові Java, і які класи реалізують ці компоненти? Дайте коротку характеристику кожного класу.
40. Як створюється рядок меню і як додаються меню в рядок меню в програмі на мові Java?
41. Як додаються пункти меню і підміню в меню
42. Як додати роздільники і «гарячі» клавіші в пункти меню
43. Як додаються пункти меню з прапорцем у програмі на мові Java?
44. Як обробляються події, пов'язані з вибором меню і пунктів меню в програмі на мові Java?
45. Які типи рамок реалізуються за допомогою методів класу BorderFactory в Swing?
46. Як в Swing можна створити власну рамку?
47. Для яких цілей використовуються менеджери компонування в мові Java і як вони реалізуються?
48. Які основні менеджери компонування реалізовані в Java? Дайте коротку характеристику кожного менеджера компоновки.
49. Як в програмі на мові Java задається використовуваний менеджер компонування? Як задаються проміжки між кордоном контейнера і містяться в ньому компонент?
50. Як виконує абсолютне позиціонування компонент в графічному додатку або апплете Java?
51. Яка стратегія компонування в менеджері FlowLayout? Як задається менеджер FlowLayout, і які методи визначені для класу FlowLayout?
52. Яка стратегія компонування в менеджері GridLayout? Як задається менеджер GridLayout, і які методи визначені для класу GridLayout?
53. Яка стратегія компонування в менеджері BorderLayout? Як задається менеджер BorderLayout, і які методи визначені для класу BorderLayout?
54. Яка стратегія компонування менеджера BoxLayout і як вона реалізована в Swing?
55. Яка стратегія компонування менеджера SpringLayout і як вона реалізована в Swing?
56. Як задаються обмежувачі в менеджері компонування SpringLayout?

57. Яка стратегія компонування менеджера GroupLayout і як вона реалізована в Swing?
58. Як працює модель делегування подій в мові Java 2?
59. Як організована ієрархія класів подій в мові Java?
60. Які типи подій визначені в мові Java? Дайте коротку характеристику кожного типу події.
61. Які змінні і методи визначені для класу ComponentEvent і інтерфейсу ComponentListener в мові Java?
62. Які змінні і методи визначені для класу ContainerEvent і інтерфейсу ContainerListener в мові Java?
63. Які змінні і методи визначені для класу FocusEvent і інтерфейсу FocusListener в мові Java?
64. Які змінні і методи визначені для класів і інтерфейсів, пов'язаних з обробкою вікна в мові Java?
65. Які змінні і методи визначені для класу обробки подій введення InputEvent в мові Java?
66. Які змінні і методи визначені для класу MouseEvent і інтерфейсів, пов'язаних з обробкою подій миші в мові Java?
67. Які змінні і методи визначені для класу MouseWheelEvent і інтерфейсу MouseWheelListener в мові Java?
68. Які кроки необхідно виконати в програмі на мові Java для реалізації обробки події?

При розробці методичних вкзівок використовувався матеріал автора Шонін В.А.

Електронне видання

МЕТОДИЧНІ ВКАЗІВКИ
для виконання лабораторних робіт на тему
«Використання системи Swing в Java»
із дисциплін
«Програмування мобільних комп'ютерних систем»,
«Прикладне програмування в телекомунікаційних системах»,
«Програмування мобільних пристроїв телекомунікацій»
для студентів спеціальностей
6.171 "Електроніка",
6.172 "Телекомунікації та радіотехніка"
денної форми навчання

Відповідальний за випуск А. С. Опанасюк
Редактор Н. М. Мажура
Комп’ютерне верстання О. Є. Горячева

Формат 60х84/16. Ум. друк. арк. 2,32. Обл.-вид. арк. 2,18.

Видавець і виготовлювач
Сумський державний університет,
вул. Римського-Корсакова, 2, м. Суми, 40007
Свідоцтво суб’єкта видавничої справи ДК № 3062 від 17.12.2007.