УДК 004.451.84(045)
З973.20-018.7+З973.20-018.3

**Фабричев В. А.** д-р техн. наук
**Скрипка В. М.**

# ПОРТУВАННЯ КООРДИНАТНО РОЗРАХОВАНОГО ГРАФІЧНОГО ІНТЕРФЕЙСУ НА МОВУ ПРОГРАМУВАННЯ JAVA

Інститут комп'ютерних технологій Національного авіаційного університету

*В статті розглянуто основні етапи створення графічного інтерфейсу на мові програмування JAVA та особливості координатного розміщення елементів. Визначено взаємодію між допоміжними елементами інтерфейсу, що необхідні для правильного розташування основних елементів та елементів керування. Перелічені механізми об'єднання декількох схем розташування графічних елементів та їх необхідну взаємодію. Наведено ряд основних проблем, що виникають при портуванні інтерфейсів та шляхи вирішення цих проблем. Розглянуто програму створення координатного режиму розмітки елементів графічного інтерфейсу з встановленням реакцій на різні події та сигнали.*

Ідеологія мови програмування *Java* є такою, що всі елементи мають бути однаковими під всіма операційними системами, що підтримують віртуальну машину *Java* (*JVM*). Окремі підрозділи розробників програмного забезпечення створюють схему та будову графічного інтерфейсу користувача. В іншій, дещо схожій на *Java*, мові *C++* використовується координатна побудова геометрії елементів інтерфейсу. Тобто, для побудови різноманітних кнопок, написів та елементів керування програміст має спочатку визначити чіткі координати розташування цих графічних елементів на екрані комп'ютера. В Java цей механізм є специфічним та має бути окремо розрахований.

В *Java* є декілька стандартних схем розташування елементів інтерфейсу. Від самого початку розвитку мови програмування Java розробники вклали в неї необхідний для роботи з вікнами інструментарій. Так, в першій версії був представлений пакет *Abstract Windowing Toolkit*, або скорочено *AWT*. В *AWT* використовувались прості компоненти, тому він був відносно малим за розміром та легким у використанні. Другою основою реалізації були класи *Swing*. *Swing* це повнофункціональний інструментарій для розробки графічного інтерфейсу користувача. Саме тому зараз більшість програм починається з імпортування *java.awt* та *javax.swing*. *AWT* містив у собі простий компонент фрейму, що дозволяв добавляти компоненти відразу у нього, але "правильні" програми завжди містили в собі панель на якій потім розташовувались елементи. Клас *JFrame* бібліотеки *Swing* є більш складним. Він має два вже створених у ньому, а не один, контейнери. *ContentPane* є головним контейнером. Краще використовувати його як головний контейнер об'єкту *JFrame*. Компонент *GlassPane* має прозорий фон та розташований поверх контейнеру *ContentPane*. Його головне призначення – тимчасове малювання чого-небудь поверх контейнеру *ContentPane*. Проектування розмітки вікна починається з вибору способу розташування. В *AWT* існує п'ять стандартних менеджерів розмітки (*layout manager*) (табл. 1) та декілька спеціалізованих класів в *javax.swing*.

Менеджер розмітки *FlowLayout* використовується за умовчуванням в класах *JPanel*, *Applet* та *JApplet*. Він розміщує компоненти за направленням, що відповідає локалізації програмного забезпечення. Тобто, зліва направо в європейських та англомовних странах, та справа наліво для арабських чи івриті. Загальний набір елементів центрується в середині вікна. Для класів *JFrame* та *JWindow* за умовчуванням використовується менеджер розмітки *BorderLayout*. Він ділить екран на п'ять областей, що відповідають направленням світу. *North* – північ, *South* – південь, *West* – захід, *East* – схід та *Center* – центральна частина. Схематичну розмітку цього менеджера розмітки приведено на рис. 1. Та-

кож потрібно відмітити, що кожен компонент має метод під назвою *getPreferredSize*(), який менеджери розмітки використовують для визначення де і як розміщувати елементи. Правильно написаний компонент переопріділяє цей метод та робить постійну корекцію при зміні розмірів інтерфейсу. Наприклад, кнопка буде вказувати, що вана повинна бути достатньо великою, тому що кнопка або мітка буде містити в собі її текст чи знак та їхні відступи. Так, використовуючи таку методу написання всіх елементів інтерфейсу, можливо встановити розмір фрейму (об'єкту класу *Jframe*) за схемою: «встановити розмір, що складається з суми всіх розмірів елементів інтерфейсу з потрібними відступами з усіх сторін». Це робиться єдиним виловом методу pack(), що не отримує жодного аргументу. Цей метод проводить опитування всіх компонентів, що встроєно, дізнаючись про бажаний для кожного компоненту розмір, а метод *getPreferredSize*() кожного вкладеного контейнеру проведе туж саму операцію зі своєми компонентами. Потім для об'єкту JFrame буде встановлено найбільш придатний розмір для придання компонентам бажаного розміру. Якщо не використовувати метод *pack*(), то потрібно скористатись методом *setSize*(), що потребує або вказання ширини та висоти фрейму, або передачі об'єкту класу *Dimension*, що містить в собі цю інформацію.

Всі менеджери, що були перелічені, не використовують вкладки, отже, не мають механізмів керування ними. Для вкладок існує спеціалізований клас *JTabbedPane*, що діє як комбінований контейнер та менеджер розмітки. Він реалізує традиційну розмітку з використанням вкладок. Для того, щоб добавити до розмітки вкладку, не потрібно використовувати метод *setLayout*(), а потрібно створити об'єкт *JTabbedPane* та визвати його метод *addTab*() з параметрами *String* та *Component* (Лістинг 1).

Для багатьох програмістів, що переходять до програмування на *Java* з інших мов, особливо схожих на *C++*, є незручним або неможливим використання стандартних менеджерів розмітки замість звичного координатного. Розглянемо, як можна використати координатну розмітку в створенні графічного інтерфейсу *Java*. Створимо невеликий графічний інтерфейс, що є звичним для багатьох програм з нескладним керуванням, готовий вигляд якого приведений на рис. 2. Для того, щоб не прив'язуватись до однієї конкретної мови програмування, з якої проводиться портування, припустимо, що вигляд інтерфейсу та його функціональність вже визначені.

Алгоритм створення розмітки простий, але для повного розуміння наведено ще раз:

1. Визначення максимальних розмірів вікна.
2. Перевірка на здатність відображення вікна потрібного розміру.
3. Виходячи з визначених в попередніх пунктах величин, розрахування геометрично пропорційних координат елементів інтерфейсу.
4. Послідовне зображення вже розрахованих елементів графічного інтерфейсу.
5. Встановлення необхідних реакцій на події графічного інтерфейсу, що спричиняють зміну його вигляду.

Розглянемо програму, що створює приведений графічний інтерфейс.

Першим етапом має бути визначення необхідних розмірів з урахуванням графічного режиму екрану. Наприклад, таким чином ми визначаємо максимальний розмір для фрейму (Лістинг 2 ).

Другим етапом задаються потрібні розміри для клавіш, розраховуються інші елементи графічного інтерфейсу.

Третім етапом встановлюється режим відображення елементів інтерфейсу за координатною схемою. Цей етап є дуже важливий, тому розглянемо його детально.

Спочатку встановлюється невизначений менеджер розмітки.

*getContentPane*().*setLayout*(*null*);

Цим ми задали параметр використання абсолютних координат екрану, що

дасть можливість самостійного встановлення координат елементів. Для прикладу, встановлюється максимальний розмір фрейму.

*setSize(FsizeX, FsizeY);*

Фрейм не повинен відображатись до моменту його закінченого конструювання тому його треба зробити невидимим.

*setVisible(false);*

Тепер черга за створенням необхідних елементів на поверхні фрейму.

Для контролю над помилками виконання методи створення елементів виконується в конструкції *try-catch.*

```
try {
  Init();
}
catch (Exception e) {
  e.printStackTrace();
}
```

Наприкінці – відображення всього зробленого.

*setVisible(true);*

Так як динамічний перерахунок координат відсутній, то зміна розміру фрейма має бути заборонена.

*this.setResizable(false);*

Окрім створення елементів інтерфейсу необхідне встановлення потрібних реакцій за схемою, що приводится: (Лістинг 3)

Головну функцію програми треба використати для ініціалізації інтерфейсу та встановлення механізмів обробки закриття програми. При закритті програми, *JVM* сама закриє майже всі незакриті для використання ресурси. Але послідовність, з якою вона буде це робити, може не запи-сати результатів роботи, тому краще самим встановити метод обробки закриття фрейму (Лістинг 4).

Мова програмування *Java* дозволяє використовувати декілька менеджерів водночас. Таким чином, комбінуючи їх, можна створювати графічні інтерфейси будь-якої складності та функціональності. Найбільш функціональною є менеджер розмітки, що дозволяє створення сітки, відносно котрої ведеться розмітка. Для створення координат, таких як використовує мова програмування *C++*, потрібно послідовно відображати однотипні елементи, що знаходяться не перетині координатних вісей. Переносячи вісь, ми одночасно переносимо місце, де буде відтворюватися елемент інтерфейсу. На цьому місці може бути розташований ще один або декілька менеджерів розмітки, в тому числі і той, що ми використовували для координатного створення інтерфейсу. Ця схема дозволяє без великих зусиль створити первинний зовнішній вигляд, але потім потрібно встановити реакції на зміну розмірів фрейму, екрану, шрифтів, та ін. Це тягне за собою велику кількість нових розрахунків та роботи по синхронізації дій по відображенню елементів. Ця робота може виявитись складнішою, ніж первісне відтворення, тому рекомендується у мові програмування Java уникати координатної розмітки з абсолютним завданням координат та використовувати відносні, що постійно динамічно оновлюються та корегуються менеджерами розмітки.

*Таблиця 1*

| Назва | Примітка |
|---|---|
| *FlowLayout* | Послідовне розміщення в контейнері |
| *GridLayout* | Звичайна сітка, всі елементи одного розміру |
| *CardLayout* | Виводить в даний момент часу один з багатьох компонентів. Використовується для створення програм в стилі програми-мастера (*wizard*) |
| *GridBagLayout* | Гнучка система розмітки, але сама складна |
| *BorderLayout* | Має п'ять географічних областей |

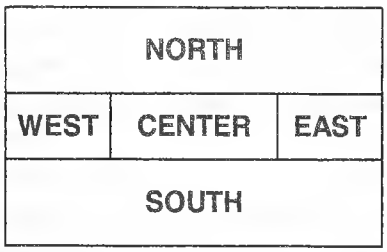


Рис. 1. П'ять областей менеджера розмітки *BorderLayout*

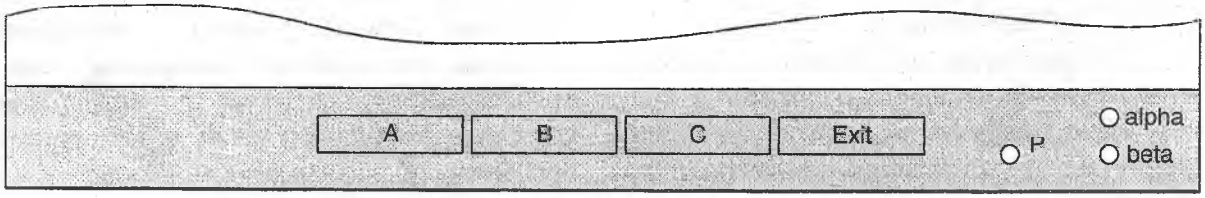


Рис. 2. Зовнішній вигляд спрощеного інтерфейсу

**Лістинг 1**

```
public TBD {
  TabPane=new JTabbedPane();
  TabPane.add(new Jlabel ("One", Jlabel.
  CENTER)."First  pane");
  TabPane.add(new Jlabel ("Two", Jlabel.
   CENTER). "Second pane");
        }
...
getContentPane().add(new TBD(). TabPane);
...
```

**Лістинг 2**

```
FsizeY = (int) Toolkit. getDefaultToolkit().
getScreenSize().getHeight();
FsizeX = (int) Toolkit. getDefaultToolkit().
getScreenSize().getWidth();
```

**Лістинг 3**

```
  ButtonA = new JButton("A");
  getContentPane().add(ButtonA);
  ButtonA.setLocation(X,Y);
  ButtonA.setSize(BsizeX, BsizeY);
  ButtonA.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed (ActionEvent) {
      ButtonC.setEnabled(false);
      ButtonB.setEnabled(false);
        }
      });
```

**Лістинг 4**

```
cInterface1.addWindowListener(new
WindowAdapter() {
  public void windowClosing
(WindowEvent event) {
    JFrame fr = (JFrame)event. getSource();
    fr.setVisible(false);
    fr.dispose();
    System.exit(0);
     }
    });
```

**Список літератури**

1. *Дарвин Ян. Ф.,* Java. Сборник рецептов для профессионалов. – С.Пб.: Питер, 2002. – 768 с.

2. *SUN* microsystems, JDK documentation.

3. *Джошуа Блох.* Java. Эффективное программирование. – М: Лори, 2002. – 240 с.

4. *Кей С. Хорстманн, Гари Корнелл.* Библиотека профессионала. Java 2. Т. 2. Тонкости программирования. – М: Вильямс, 2002. – 1120 с.