Начинаем программировать на языке Java. Часть 4

Дмитрий Рамодин

Сегодня мы рассмотрим самый большой и, наверное, самый полезный раздел языка Java, связанный с реализацией пользовательского интерфейса. Для этого мы изучим базовые классы упаковки java.awt (Abstract Window Toolkit).

Итак, что же такое awt? Это набор классов Java, каждый из которых отвечает за реализацию функций и отображение того или иного элемента графического интерфейса пользователя (GUI). Практически все классы визуальных компонентов являются потомками абстрактного класса Component. Лишь визуальные элементы меню наследуются от другого класса — MenuComponent. Управляющие элементы представлены следующими классами: Button (кнопка), Checkbox (кнопка с независимой фиксацией), Choice (раскрывающийся список Windows), Label (строка), List (список выбора Windows) и Scrollbar (полоса прокрутки). Это достаточно простые классы, наследуемые от абстрактного класса Component напрямую.

Однако в составе java.awt имеются классы интерфейсных элементов, имеющие промежуточного предка. Хорошим примером тому является класс Panel для создания различных панелей. У него имеется промежуточный абстрактный класс-предок Container, служащий родоначальником многих классов-контейнеров, способных содержать в себе другие элементы интерфейса. От этого же класса наследуется класс окна Window, представляющий на экране простейшее окно без меню и рамки. У этого класса есть два часто используемых потомка: Dialog, название которого говорит само за себя, и Frame — стандартное окно Windows. Еще один промежуточный класс TextComponent порождает два полезнейших в работе класса - TextField (аналог строки ввода Windows) и многострочное окно текстового ввода TextArea. Особняком от всех элементов стоит класс Canvas. Его визуальное представление — пустой квадрат, на котором можно выполнять рисование и который может обрабатывать события нажатия

кнопок мыши.

От своего предка Component все визуальные элементы перенимают общее для них всех поведение, связанное с их визуальной и функциональной сторонами. Вот список основных, выполняемых компонентами, функций и методов для их реализации:

изменение шрифта - методы getFont, setFont, getFontMetrics;

изменение цвета шрифта - методы setForeground(Color) и getForeground() — для установки и чтения цвета самого шрифта, а также setBackground(Color) и getBackground() для установки, а также чтения цвета фона, на котором отображается текст;

размер и позиция на экране - методы preferredSize() и minimumSize() сообщают менеджеру раскладок о предпочтительном и минимальном размерах компонента, соответственно;

отображение компонента - методы paint(), update() и repaint();

обработка сообщений - методы handleEvent(), action(), keyDown(), keyUp(), mouseDown(), mouseUp(), mouseDrag(), mouseMove(), mouseEnter() и mouseExit().

1. Контейнеры

Любой компонент, требующий показа на экране, должен быть добавлен в класс-контейнер. Контейнеры служат хранилищем для визульных компонентов интерфейса и других контейнеров. Простейший пример контейнера — класс Frame, объекты которого отображаются на экране как стандартные окна с рамкой.

Чтобы показать компонент пользовательского интерфейса в окне, требуется создать объект-контейнер, например окно класса Frame, создать требуемый компонент и добавить его в контейнер, а уже затем отобразить его на экране. Несмотря на столь длинный список действий, в исходном тексте этот процесс занимает всего несколько строк:

```
// Создается текстовый объект
с надписью "Строка"
Label text = new Label("Строка");
```

```
// Объект добавляется в некий контейнер
SomeContainer.add (text);
// Отображается контейнер
SomeContainer.Show();
...
```

Все достаточно просто. Лишь поясним метод add(). Этим методом и производится собственно добавление элемента интерфейса в окно контейнера. В нашем примере мы использовали самый простой вариант этого метода, принимающий единственный аргумент-ссылку на вставляемый объект. Но есть еще два варианта метода add() с двумя аргументами. В первом из них передаются порядковый номер в списке управляющих элементов контейнера, куда будет вставлен добавляемый элемент, и ссылка на вставляемый объект. Во втором варианте первый аргумент — строка, указывающая место в окне, где должен быть размещен вставляемый объект интерфейса, а второй — ссылка на вставляемый объект. Строк, допустимых в качестве первого аргумента, всего пять: North, South, East, West и Center. Подробнее мы рассмотрим их в разделе, посвященном раскладкам.

```
// Вставить элемент в окно контейнера
add(someControl);
// Вставить элемент после других
// элементов в контейнере
add(-1, someControl);
// Вставить элемент в окно контейнера
// у его верхней границы
add("North", someControl);
```

Само собой разумеется, коли есть методы для добавления визуальных элементов, имеются и противоположные им методы, удаляющие элементы из окна контейнера. Их два: метод удаления конкретного элемента remove(), принимающий в качестве параметра ссылку на удаляемый объект, и метод удаления всех визуальных компонентов removeAll().

2. Использование компонентов интерфейса

Теперь рассмотрим ключевые моменты в использовании классов каждого компонента в отдельности. Отметим, что здесь не приводятся способы обработки сообщений от

компонентов, поскольку этим мы будем заниматься в наших практических занятиях.

2.1. Label

С помощью класса Label можно создавать текстовые строки в окне Java-программ и аплетов. По умолчанию текст будет выровнен влево, но, используя метод setAlignment с параметрами Label.CENTER и Label.RIGHT, можно выровнять строку по центру или по правому краю. Вы можете задать выводимый текст либо при создании объекта класса Label, либо создать пустой объект и уже затем определить его текст вызовом метода setText(). То же касается и выравнивания, для управления которым имеется метод setAlignment(). Вот примеры использования трех различных конструкторов для объекта класса Label:

```
Label first = new Label();
Label second =
new Label("Some text");
Label third =
new Label("Some text", Label.CENTER);
```

Вы можете узнать о текущем тексте и его выравнивании, вызвав методы getText() и getAlignment() соответственно.

2.2. Button

Класс Button представляет на экране кнопку. У этого класса имеется два типа конструктора. Первый из них создает кнопку без надписи, второй — с надписью:

```
Label first = new Label();
Label second =
new Label("Some text");
Label third =
new Label("Some text", Label.CENTER);
```

В любой момент можно создать или изменить надпись на кнопке, вызвав метод setLabel() и передав ему в качестве аргумента строку, которая будет написана на кнопке, а вызывая методы disable() и enable(), кнопку можно отключать (т.е. запрещать обработку нажатий) и включать.

2.3. Checkbox

Класс Checkbox отвечает за создание и отображение кнопок с независимой фиксацией. Это кнопки, имеющие два состояния: "включено" и "выключено". Щелчок на такой кнопке приводит к тому, что ее состояние меняется на противоположное. Если разместить несколько кнопок с независимой фиксацией внутри элемента класса CheckboxGroup, то вместо них мы получим кнопки с зависимой фиксацией, то есть группу кнопок, среди которых в один и тот же момент может быть включена только одна. Если нажать какую-либо кнопку из группы, то ранее нажатая кнопка будет отпущена. Вы наверняка сталкивались с такими группами в различных Windows-программах.

Ниже приведены фрагменты, иллюстрирующие создание как независимых кнопок, так и кнопок с независимой фиксацией:

```
// Создать две независимые отмечаемые
// кнопки
Checkbox first, second;
first = new Checkbox("First checkbox");
first.setState(true);
// Включить кнопку
second = new Checkbox("Second
checkbox");
...
// Создать две связанные радиокнопки
Checkbox first, second;
CheckboxGroup group = new CheckboxGroup();
first = new Checkbox
//true — кнопка включена
("First radiobutton", group, true);
second = new Checkbox
//false — кнопка выключена
("Second radiobutton", group, false;
```

Когда потребуется программно установить активную кнопку, можно вызвать метод setCurrent(). Для определения активной в настоящий момент кнопки, вызывайте метод getCurrent().

2.4. Choice

Когда требуется создать раскрывающийся список, можно прибегнуть к помощи класса Choice. Создать его достаточно просто. К примеру, так выглядит реализация списка из трех пунктов:

```
Choice choice = new Choice();
choice.addItem("First");
choice.addItem("Second");
choice.addItem("Third");
```

Полезные методы класса Choice:

- countItems() — считать количество пунктов в списке;
- getItem(int) — возвратить строку с определенным номером в списке;
- select(int) — выбрать строку с определенным номером;
- select(String) — выбрать определенную строку текста из списка.

2.5. List

Класс List (список) по назначению очень похож на класс Choice, но предоставляет пользователю не раскрывающийся список, а окно с полосами прокрутки, внутри которого находятся пункты выбора. Любой из этих пунктов можно выбрать двойным щелчком мыши или, указав на него мышью, нажать клавишу <Enter>. Причем можно сделать так, что станет возможным выбор нескольких пунктов одновременно.

Создание объекта класса List может происходить двумя способами. Вы можете создать пустой список и добавлять в него пункты, вызывая метод addItem. При этом размер списка будет расти при добавлении пунктов. Другой способ позволяет сразу ограничить количество видимых в окне списка пунктов. Остальные пункты выбора можно увидеть, прокрутив список. Вот пример для обоих способов:

```
// Создание списка без ограничения
// размера
List list1 = new List();
list.addItem("First");
list.addItem("Second");
```

```
list.addItem("Third");
list.addItem("Forth");
// Создание списка, ограниченного
// размером в два пункта выбора
List list2 = new List(2,true);
list.addItem("First");
list.addItem("Second");
list.addItem("Third");
list.addItem("Forth");
```

Полезные методы класса List:

- getItem(int) — считать текст пункта выбора;
- countItems() — посчитать количество пунктов выбора в списке;
- replaceItem(String, int) — заменить элемент выбора в указанной позиции;
- clear() — очистить список;
- delItem(int) — убрать из списка определенный пункт;
- delItems(int, int) — убрать элементы выбора с номерами, входящими в интервал от номера, указанного первым параметром, до номера, указанного вторым параметром;
- getSelectedIndex() — узнать порядковый номер выделенного пункта; если возвращается -1, то выбрано несколько пунктов;
- getSelectedIndexes() — возвратить массив индексов выделенных пунктов;
- getSelectedItem() — прочитать текст выделенного пункта выбора;
- getSelectedItems() — возвратить массив строк текста выделенных пунктов;
- select(int) — выделить пункт с определенным номером;
- deselect(int) — снять выделение с определенного пункта;
- isSelected(int) — возвратить значение true, если пункт с указанным номером выделен, иначе возвратить false;
- getRows() — возвратить количество видимых в списке строк выбора;
- allowsMultipleSelections() — возвратить true, если список позволяет множественный выбор;
- setMultipleSelections() — включить режим разрешения множественного выбора;
- makeVisible(int) — сделать элемент с определенным номером видимым в окне списка;
- getVisibleIndex() — возратить индекс элемента выбора, который последним после вызова метода makeVisible() стал видимым в окне списка.

2.6. Scrollbar

Класс Scrollbar представляет на экране знакомую всем полосу прокрутки. С помощью этого элемента можно прокручивать изображение и текст в окне либо устанавливать некоторые значения. Чтобы создать полосу прокрутки, необходимо вызвать конструктор объекта класса Scrollbar. Это можно сделать тремя способами:

```
// Создать полосу прокрутки
// с параметрами по умолчанию
new Scrollbar();
// Создать полосу прокрутки
// с ориентацией,в данном случае -
// вертикальной
new Scrollbar(Scrollbar.VERTICAL);
```

Третий конструктор создает объект класса Scrollbar, сразу задавая все необходимые параметры:

```
new Scrollbar
     ( <ориентация>,
     <текущее значение>,
     <видно>, <мин. значение>,
     <макс. значение>);
```

где

<ориентация> — ориентация полосы, задаваемая константами Scrollbar.HORIZONTAL и Scrollbar.VERTICAL;

<текущее значение> -начальное значение, в которое помещается движок полосы прокрутки;

<видно> — сколько пикселов прокручиваемой области видно, и насколько эта область будет прокручена при щелчке мышью на полосе прокрутки;

<мин. значение> — минимальная координата полосы прокрутки;

<макс. значение> — максимальная координата полосы прокрутки.

Обычно в качестве прокручиваемой области выступает объект класса Canvas или

порожденный от него объект. При создании такого класса его конструктору необходимо передать ссылки на полосы прокрутки.

2.7. TextField и TextArea

Два родственных класса, TextField и TextArea позволяют отображать текст с возможностью его выделения и редактирования. По своей сути это маленькие редакторы: однострочный (TextField) и многострочный (TextArea). Создать объекты этих классов очень просто: нужно лишь передать размер в символах для класса TextField и размер в количестве строк и символов для класса TextArea:

```
TextField tf = new TextField(50);
TextArea ta = new TextArea(5, 30);
```

Можно запретить редактировние текста в окне:

```
tf.setEditable(false);
ta.setEditable(false);
```

Полезные методы классов TextField и TextArea:

- getText() — считать текст;
- setText() — отобразить текст;
- selectAll() — выделить весь текст;
- getSelectedText() — считать выделенный текст;
- isEditable() — проверить, разрешено ли редактирование текста;
- getSelectionStart() — возвратить начало выделения;
- getSelectionEnd() — возвратить окончание выделения;
- select() — выделить текст между начальной и конечной позициями;
- getColumns() — возвратить количество символов в строке редактирования.

В свою очередь, класс TextField имеет дополнительные методы:

- setEchoChar() — установить символ маски; применяется при введении паролей;
- char getEchoChar() — узнать символ маски;
- echoCharIsSet() — узнать, установлен ли символ маски.

Для класса TextArea добавляются другие методы:

- int getRows() — считать количество строк в окне;

- insertText(String, int) — вставить текст в определенной позиции;
- replaceText(String, int, int) — заменить текст между заданными начальной и конечной позициями.

2.8. Panel

Класс Panel (панель) — это простой контейнер, в который могут быть добавлены другие контейнеры или элементы интерфейса. Обычно он используется в тех случаях, когда необходимо выполнить сложное размещение элементов в окне Java-программы и аплета.

Создание объекта класса Panel элементарно:

```
Panel p1 = new Panel();
```

Как и в случае с другими контейнерами, элементы интерфейса могут быть добавлены методом add().

2.9. Frame

Одним из самых важных классов пользовательского интерфейса можно считать класс Frame. С его помощью реализуются окна для Java-программ и аплетов. В отличие от других классов пользовательского интерфейса, экземпляры класса Frame создаются редко. Обычно от него наследуется новый класс, а уже затем создается экземпляр нового класса:

```
public class NewWindow extends Frame
{
  TextArea output;
  public NewWindow (String title)
   { super(title); }
...
   public static void main (String args[])
  {
// Создание экземпляра нового класса
  NewWindow win =
  new NewWindow("New Window Class");
// Показать его на экране
```

```
win.show();
        }
}
```

Полезные методы класса Frame:

- pack() — изменить размер компонентов в окне так, чтобы их размер был максимально приближен к желаемому;
- getTitle() — возвратить заголовок окна;
- setTitle(String) — установить заголовок окна;
- getIconImage() — возвратить пиктограмму окна;
- setIconImage(Image) — установить пиктограмму окна;
- getMenuBar() — возвратить объект меню окна;
- setMenuBar(MenuBar) — установить меню окна;
- remove(MenuComponent) — убрать определенный компонент из меню окна;
- isResizable() — возвратить true, если размер окна можно изменять, иначе — false;
- setResizable(boolean) — разрешить изменение размеров окна;
- getCursorType() — возвратить текущий тип курсора мыши для окна;
- setCursor(int) — установить тип курсора мыши для окна.

2.10. Dialog

Для поддержания связи с пользователем применяется класс Dialog, на основе которого можно создавать диалоговые панели. В отличие от простых окон диалоговые панели зависят от того или иного окна, и поэтому в их конструкторах присутствует параметр-ссылка на окно класса Frame, владеющее этой диалоговой панелью. Как и в случае с классом Frame, класс Dialog сам по себе практически не применяется. Обычно от него наследуется новый класс, экземпляр которого и создается:

class NewDialog extends Dialog

```
class NewDialog extends Dialog
{
...
   NewDialog(Frame frame, String title)
   { super(dw, title, false); }
...
}
```

Поскольку диалоговые панели могут быть модальными (блокирующими работу с другими окнами) и немодальными, в конструкторах класса Dialog последний параметр определяет модальность. Если он равен true, то диалоговое окно создается модальным, в противном случае оно позволяет переключиться на другое окно приложения.

Помимо общих для всех окон методов getTitle(), setTitle(), isResizable() и setResizable() у класса Dialog имеется метод isModal(), возвращающий true, если диалоговая панель модальна.

3. Классы элементов меню

Едва ли какое-то современное приложение сможет обойтись без полосы меню в окне. Поэтому в языке Java имеются сразу несколько классов для создания меню, унаследованных от класса MenuComponent. Первый из них, MenuBar, это основной класс всей системы меню, служащий контейнером для других классов. Когда вы создаете окно, то в качестве ссылки на добавляемое меню нужно передать ссылку на класс MenuBar.

Следующий класс Menu на полосе меню отображается как пункт выбора, который, если по нему щелкнуть, раскрывается в виде странички с пунктами выбора (pop-up menu). Сами же элемены выбора меню обычно реализуются как экземпляры классов MenuItem (простой элемент выбора) и CheckboxMenuItem (отмечаемый элемент выбора). Взгляните на пример создания полнофункциональной полосы меню:

```
// Добавление полосы меню в окно класса Frame
public class NewWindow extends Frame
{
    public NewWindow()
    {
// Создаем полосу меню
   MenuBar menuBar = new MenuBar();
// Создаем первое меню
// Второй аргумент true говорит
// о том, что меню отрываемое.
// Эта опция пока не работает
// в Windows
    Menu menu1 =
```

```
    new Menu("Menu 1", true);
    menuBar.add(menu1);
// Создать и добавить первый пункт
// первого меню
// Это обычный элемент меню
    MenuItem item1_1 =
    new MenuItem("Item #1");
    menu1.add(item1_1);
// Создать и добавить второй пункт
// первого меню
// Это отмечаемый элемент меню
    CheckboxMenuItem item1_2 =
    CheckboxMenuItem("Item #2");
    menu1.add(item1_2);
// Создать и добавить второе меню
    Menu menu2 = new Menu("Menu 2");
    menuBar.add(menu2);
// Создать и добавить меню
// следующего уровня
    Menu nextLevel =
    New Menu("Next Level Menu");
    menu2.add(nextLevel);
    }
...
}
```

Как видите, создание меню хотя и муторный, но вовсе не сложный процесс. Если вы обратили внимание, то во второе меню добавляется не пункт выбора класса MenuItem, а меню класса Menu. Это приводит к тому, что при нажатии на пункт 2 полосы меню рядом появляется следующее меню, выбрав из которого Next Level Menu получили очередное меню. Таким способом в Java реализовано каскадирование меню.

4. Canvas

Если вам требуется создать область для вывода данных (изображений и текста) или рисования, то необходимо воспользоваться классом Canvas. Это виртуальный "холст" для художника и писателя. Класс Canvas также можно использовать как базовый класс для создания своих элементов интерфейса, например собственной кнопки.

Создание любого наследника от Canvas сводится к реализации методов paint() для прорисовки изображения и текста на рабочей поверхности и методов minimumSize() и preferredSize(), управляющих ее размерами.

5. Раскладки

Остался один вопрос, которого мы с вами не коснулись, — раскладки компонентов. Для того чтобы управлять расположением элементов внутри окон-контейнеров, в Java существует менеджер раскладок (layout manager). От него наследуются пять классов, определяющих тот или иной тип расположения компонентов пользовательского интерфейса в окне. Когда вам требуется изменить тип расположения, вы создаете тот или иной класс раскладки, отвечающий вашим запросам, и передаете его в вызываемый метод setLayout(), изменяющий текущую раскладку:

```
// Установить расположение элементов
// вдоль рамки окна
setLayout( new BorderLayout() );
```

Дадим краткую характеристику классам-раскладкам.

FlowLayout. Это простейший способ расположения элементов один за другим, применяемый по умолчанию. Когда в одной строке уже не помещаются новые элементы, заполнение продолжается с новой строки.

CardLayout. При раскладке этого типа элементы размещаются один за другим, как карты в колоде. Обычно такой расклад удобен, если вам необходимо динамически изменять интерфейс окна. Кроме того, вы можете делать элементы, находящиеся один над другим по очереди.

BorderLayout. Эта раскладка размещает элементы либо рядом с выбранным краем окна, либо в центре. Для этого после установки BorderLayout добавление элементов в окно-контейнер производится методом add() с дополнительным параметром, задаваемым строками North, South, East, West и Center. Каждая из них означает тот край окна, к которому необходимо прижать вставляемый элемент.

GridLayout. GridLayout располагает элементы один за другим внутри некоторой условной таблицы. Все элементы будут одинакового размера. Размер ячеек можно

программно изменять.

GridBagLayout. Это самая мудреная, но в то же время и самая мощная раскладка. Она располагает элементы в условной таблице, как это делается в случае с GridLayout. Но в отличие от последней, можно варьировать размер каждого элемента в отдельности. Правда, придется набрать дополнительно не одну строчку исходного текста.

Это было последнее теоретическое занятие. Все дальнейшие уроки будут носить уже практический характер. Так что запаситесь терпением и подходящим инструментом для программирования на Java. Подойдет любой из следующих пакетов: Microsoft Visual J++, Symantec Cafe, Java Add-On из состава Borland C++ 5.0 или Sun Java WorkShop. Если желание пользоваться командными файлами у вас еще не прошло, то можно загрузить с Web-сервера http://java.sun.com "родной" вариант компилятора Java компании Sun — Java Development Kit (JDK).

Мир ПК #03/97