**ЛАБОРАТОРНАЯ РАБОТА №3**

**РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА  
С ИСПОЛЬЗОВАНИЕМ ФРЕЙМОВ, ЭЛЕМЕНТОВ УПРАВЛЕНИЯ**

**Цель:** научиться создавать простейшие GUI-приложения.

**Задание для самостоятельного выполнения:**

**Задание I (одно на выбор)**

1. Разработать приложение управления тремя списками, расположенны­ми горизонтально. Приложение должно обеспечивать перемещение любого вы­бранного элемента или содержимого всего списка в следующий список по часовой стрелке: из первого - во второй, из второго в третий, из третьего в пер­вый. Элемент при перемещении должен исчезать из одного списка и появ­ляться в другом. Приложение должно обеспечивать управление вторым списком - добавление нового элемента, редактирование, удаление.

1. Разработать приложение управления тремя списками, расположенны­ми горизонтально. Приложение должно обеспечивать перемещение выбранного элемента из первого во второй, из второго в третий, из третьего в первый спи­сок и наоборот. Направление перемещения элемента из списка в список должно определяться выбором из набора флажков *(CheckboxGroup).* Элемент при пе­ремещении должен исчезать из одного списка и появляться в другом. Помимо того, приложение должно обеспечивать управление всеми списками - добавле­ние нового элемента, редактирование, удаление.
2. Разработать приложение, обеспечивающее возможность множествен­ного выбора элементов из списка. Выбранные элементы должны образовывать строку текста и помещаться в текстовое поле. Предусмотреть возможность вы­вода сообщения в диалоговое окно *(Dialog)* в случае, если суммарное количе­ство символов будет превышать 100.

**Задание II (одно на выбор)**

1. Разработать приложение управления тремя списками, расположенны­ми на диалоге горизонтально. Приложение должно обеспечивать перемещение некоторого (указанного в наборе флажков *(CheckboxGroup))* количества вы­бранных элементов из списка в список. Перемещение элементов осуществлять слева направо. Элемент при перемещении не исчезает, а выделяется. Помимо этого, приложение должно обеспечивать заполнение помеченного флажком списка 10 строками. Предусмотреть очистку помеченного списка.
2. Разработать приложение управления списком. Вывести два флажка *(Checkbox).* При первом включенном флажке осуществляется выбор всех не четных строк, при втором включенном флажке осуществляется выбор всех чет­ных строк и перенос их в раскрывающийся список *(Choice).*
3. Разработать приложение управления списком. Вывести два флажка *(Checkbox).* При первом включенном флажке осуществляется выбор всех не­четных строк и их удаление, при втором включенном флажке осуществляется выбор всех четных строк и перенос их во второй список. Предусмотреть обнов­ление элементов списка и очистку второго списка.

**Задание III (одно на выбор)**

1. Разработать приложение, обеспечивающее поиск в двух раскрываю­щихся списках *(Choice)* фрагмента текста. Набором флажков *(CheckboxGroup)* указывать, в каком списке будет осуществляться поиск. Строки, в которых бу­дет найден искомый фрагмент, должны быть выделены (предполагается, что несколько строк могут иметь искомый фрагмент). Помимо этого приложение должно обеспечивать управление содержимым списков - добавление нового элемента, редактирование, удаление.
2. Разработать приложение, обеспечивающее возможность множест­венного выбора элементов из списка. Выбранные элементы должны образовы­вать строку текста и выводиться в соседний список. Предусмотреть возмож­ность вывода сообщения в диалоговое окно *(Dialog)* в случае, если суммарное количество символов будет превышать 100.
3. Разработать приложение управления тремя списками («Фамилия», «Имя», «Отчество») и строкой редактирования *(TextField).* Для отображения строки редактирования вызывается диалоговое окно *(Dialog).* В строку редак­тирования вводится информация в формате «Фамилия Имя Отчество». По за­вершении ввода диалоговое окно закрывается, фамилия должна появиться в списке «Фамилия», имя - в списке «Имя», отчество - в списке «Отчество». Пре­дусмотреть возможность множественного выбора фамилий и записи их в от­сортированном порядке в четвертый список.

**События**

В основу Java-программирования наряду с другими положен механизм обработки событий.

Событие - это объект, который описывает изменение состояния источни­ка (нажатие кнопки, выбор пункта меню, разворачивание, сворачивание окош­ка, нажатие клавиши и т.д.). Источник - это объект, генерирующий события. Одно и то же событие может быть значимым для одних объектов и несущест­венным для других.

В Java различают два механизма обработки событий:

1. с помощью метода *handleEvent()* *;*
2. с помощью модели делегирования событий.

Далее будем рассматривать особенности обработки событий с примене­нием второго механизма. В соответствии с моделью делегирования событий в обработке событий участвуют 2 объекта: источник *(source)* и блок прослушива­ния *(listener -* интерфейс для перехода конкретного вида события от конкретно­го компонента). Источник - объект, генерирующий событие. Блок прослуши­вания - объект, получающий уведомление о возникновении события, зарегист­рированного одним или несколькими источниками, путем вызова одного из его методов (методов блока прослушивания) для приема и обработки этих уведом­лений.

Методы обработки событий находятся в пакете *java. awt. event.*

**Классы событий**

В корне иерархии классов событий Java находится класс *EventObject,* на­ходящийся в пакете *java.util.* Класс *EventObject* содержит 2 метода:

* *Object getSource()* - возвращает источник события;
* *toStringf) -* возвращает название этого события в виде строки.

В Java определены несколько типов событий (табл. 3.1).

Таблица 3.1

Классы событий

|  |  |
| --- | --- |
| Класс событий | Описание |
| ActionEvent | Генерируется, когда нажата кнопка, дважды щелкнут элемент списка или выбран пункт меню |
| AdjustmentEvent | Генерируется при манипуляциях с полосой про­крутки |
| ComponentEvent | Генерируется, когда компонент скрыт, переме­щен, изменен в размере или становится види­мым |
| ContainerEvcnt | Генерируется, когда компонент добавлен или удален из контейнера |
| FocusEvent | Генерируется, когда компонент получает или те­ряет фокус |
| ItemEvent | Генерируется, когда помечен флажок или эле­мент списка, сделан выбор элемента в списке, выбран или отменен элемент меню с меткой |
| KeyEvent | Генерируется, когда получен ввод с клавиатуры |
| MouseEvent | Генерируется, когда объект перетащен мышью *(dragged),* перемещен *(moved),* произошел щел­чок *(clicked),* нажата *(pressed)* или отпущена *(re­leased)* кнопка мыши, указатель мыши входит или выходит в/за границы компонента |
| TextEvent | Генерируется, когда изменено значение тексто­вого поля |
| WindowEvent | Генерируется, когда окно активизировано, за­крыто, развернуто, свернуто и т.п. |

**Класс** *ActionEvent*

Определяет четыре целочисленные константы, которые можно использо­вать для идентификации любых модификаторов, связанных с событием дейст­вия: *ALT\_ MASK, CTRL\_MASK, META\_MASK* и *SHIFT\_MASK.* Кроме того, суще­ствует целочисленная константа *ACTION\_PERFORMED,* которую можно при­менять для идентификации action-события.

Имеет два конструктора:

*ActionEvent ( Object src, int type, String cmd);*

*ActionEvent ( Object src, int type, String cmd, int modifiers );*

*src -* ссылка на объект, который генерирует события (для следующих описываемых классов значение аналогичное);

*type -* тип события (для следующих описываемых классов значение ана­логичное);

*cmd-* командная строка события;

*modifiers -* указывает, какие клавиши-модификаторы были нажаты при генерации события *(Alt, Ctrl, Shift).*

Например, когда кнопка нажата, генерируется action-событие, которое имеет имя команды, равное метке или надписи на этой кнопке.

**FocusEvent**

Событие этого класса идентифицируется константой *FOCUS\_GAINED* и *FOCUS\_LOST.* Конструкторы класса:

*FocusEvent ( Component src, int type );*

*FocusEvent ( Component src, int type, boolean temporaryFlag);*

*temporaryFlag* - устанавливается как *true,* если событие фокуса времен­ное, иначе - *false.*

**ItemEvent**

Существует два типа *Item*-событий, которые определяются константами:

*DESELECTED -* пользователь отменил выбор элемента;

*SELECTED -* выбрал элемент списка.

Конструктор класса:

*ItemEvent (ItemSelectable src, int type, Object entry, int state );*

*entry -* передает конструктору элемент, который генерировал *Item-* событие;

*state -* состояние этого элемента.

Для того чтобы получить ссылку на объект *ItemSelectable,* используется метод *getItemSelectable().*

**KeyEvent**

Имеется три типа *Key*-событий, которые идентифицируются тремя константами:

*KEY\_PRESSED* - клавиша нажата;

*KEY\_RELEASED -* клавиша отпущена;

*KEY TYPED -* генерируется только при нажатии символьной клавиши.

Конструктор класса:

*Key Event ( Component src, int type, long when, int modifiers, int code);*

*when -* параметр, передающий конструктору системное время, когда была нажата клавиша (для следующего вписываемого класса значение аналогичное);

*modifiers -* параметр, указывающий, какие модификаторы были нажаты вместе с клавишей;

*code* - параметр, передающий конструктору код клавиши.

**MouseEvent**

Существует семь типов *Mouse*-событий, которые идентифицируются се­мью константами:

*MOUSE\_CLICKED* - пользователь щелкнул кнопкой мыши;

*MOUSE\_DRAGGED* - пользователь перетащил мышь;

*MOUSE\_ENTERED* - указатель мыши введен в компонент;

*MOUSE\_EXITED* - указатель мыши выведен из компонента;

*MOUSE\_MOVED -* мышь передвинута;

*MOUSE\_PRESSED -* кнопка мыши нажата;

*MOUSE\_RELEASED* - кнопка мыши освобождена.

Конструктор класса:

*MouseEvent (Component src, int type, long when, int modifiers, int x, int y, int clicks, boolean triggersPopup);*

*x, у -* координаты мыши;

*clicks -* подсчитывается количество щелчков;

*triggersPopup -* показывает, приводит ли это событие к появлению рас­крывающегося меню; если да, то значение параметра соответствует *true.*

*int getX( ); int getY(*); - методы для получения координат мыши.

**TextEvent**

*TEXT\_VALUE\_CHANGED -* событие, определяющее ввод текста пользо­вателем в текстовое поле.

Конструктор класса:

*TextEvent (Object src, int type);*

**Window Event**

Существует семь типов событий *WindowEvent:*

WINDOW\_ACTIVATED - окно активизировано;

WINDOW\_CLOSED - окно закрыто;

WINDOW\_DEACTIVATED - окно деактивизировано;

WINDOW\_DEICONIFIED - окно развернуто из пиктограммы;

WINDOW\_ICONIFIED - окно свернуто в пиктограмму;

WINDOW\_OPENED - окно открыто;

WINDOW\_CLOSING - пользователь потребовал закрытия окна.

Конструктор класса:

*WindowEvent (Window src, int type);*

Метод *Window getWindow()* возвращает *Window*-объект, который сгенери­ровал это событие.

**Элементы- источники событий**

В табл. 3.2 приведены некоторые элементы-источники событий, приме­няемые в Java, и их описание.

Таблица 3.2

Элементы-источники событий

|  |  |
| --- | --- |
| Источник событий | Описание |
| 1 | 2 |
| Button (кнопка) | Генерирует *action*-события в тот момент, когда нажимается кнопка |
| Checkbox (флажок) | Генерирует *item*-события, когда флажок устанав- ливается/сбрасывается |
| Choice (список с выбором) | Генерирует *item* -события, когда изменяется выбор элемента в списке с выбором |
| List (список) | Генерирует *action*-события, когда на элементе списка выполнен двойной щелчок (мышью). Гене­рирует item-события, когда элемент выделяется или снимается выделение |
| Menuitem (пункт меню) | Генерирует *action* -события, когда пункт меню вы­делен. Генерирует события элемента, когда пункт меню с меткой выделен или выделение отменяется |
| Scrollbar (полоса про­крутки) | Генерирует *adjustment-*события при манипуляциях с полосой прокрутки |
| TextField и TextArea (текстовое поле и теrстовая область) | Генерирует *text*-события, когда пользователь вводит символ |
| Window (окно) | Генерирует Window-события, когда окно активизи­руется, закрывается, деактивизируется, сворачивает­ся в пиктограмму, разворачивается из пиктограммы, открывается или выполняется выход из него *(quit)* |

**Интерфейсы прослушивания событий**

Модель делегирования событий содержит две части: источник событий и блоки прослушивания событий. Блоки прослушивания событий создаются пу­тем реализации одного или нескольких интерфейсов прослушивания событий. Эти интерфейсы определены в пакете *java.awt.event.* Когда событие происхо­дит, источник события вызывает соответствующий метод, определенный бло­ком прослушивания, и передает ему объект события в качестве параметра.

В табл. 3.3 приведены интерфейсы прослушивания событий и их методы. Когда класс реализует какой-нибудь из этих интерфейсов, то все методы ин­терфейса должны быть реализованы в этом классе. В случае, если среди мето­дов интерфейса вам необходимы не все, а только некоторые из них, то для ос­тальных методов в качестве реализации следует оставить пустые скобки {}.

**Классы пакета AWT Component, Window, Frame**

**Класс Component**

Абстрактный класс, инкапсулирующий все элементы визуального интер­фейса пользователя. Все управляющие компоненты окна пользователя являют­ся подклассами класса Component. В данном классе определено более 100 ме­тодов, которые отвечают за управление событиями, позиционирование, управ­ление размерами, управление цветами, перерисовку.

**Класс Window**

Создает окно верхнего уровня на рабочем столе. Он расширяется классом Frame, который и представляет интерфейсное окно, окно с меню, обрамлением, необходимое для создания графического приложения с его компонентами.

**Класс Frame**

Инкапсулирует полноценное окно, имеющее строку заголовка, строку меню, обрамление и углы, изменяющие размеры окна.

**Элементы управления Label, Button, Checkbox, Choice, List, Scoolbar**

Элемент управления - это компоненты, которые предоставляют различные способы взаимодействия с приложением (кнопки, флажки, полосы прокрутки и т.п.)

Элементы управления представлены следующими классами:

*Label -* с помощью класса *Label* можно создавать текстовые строки в окне Java-программ. По умолчанию текст будет выровнен влево, но, используя методы

setAligment (Label.CENTER);

setAligment (Label.RIGHT);

строку можно выровнять по центру и по правому краю. Можно создавать выводимый текст либо при создании объекта класса *Label,* либо создать пустой объект и уже затем определить его текст вызовом метода *setText().*

*Button -* представляет на экране кнопку. Конструктор:

Button first = new Button ();

Сделать кнопку неактивной можно методом *void disable().*

*Checkbox -* отвечает за создание и отображение кнопок с независимой фиксацией. Эти кнопки имеют два состояния: включено и выключено. Щелчок по такой кнопке приводит к тому, что ее состояние меняется на противополож­ное. Если разместить несколько кнопок с независимой фиксаций внутри эле­мента класса *CheckboxGroup,* то вместо них мы получаем кнопки с зависимой фиксацией. Для такой группы кнопок характерно то, что в один и тот же момент может быть включена только одна кнопка. Если нажать какую-либо кнопку из группы, то ранее нажатая кнопка будет отпущена.

*Choice -* создает раскрывающийся список.

Пример реализации списка из трех пунктов.

Пример 3.3

Choice choice = new Choice ();

choice.addItem ("First");

choice.addItem ("Second");

choice.addItem ("Third");

Методы класса *Choice.*

*int countItems() -* считать количество пунктов в списке;

*String getItem(int)* - возвратить строку с определенным номером в списке;

*void select(int) -* выбрать строку с определенным номером.

*List -* по назначению похож на *Choice,* но предоставляет пользователю не раскрывающийся список, а окно с полосами прокрутки. Такое окно содержит пункты выбора.

Создать объект класса *List* можно двумя способами:

1. Создать пустой список и добавить в него пункты методом *addItem().* При этом размер списка будет расти при добавлении пунктов.

Пример 3.4

List list 1 = new List ();

list 1.addItem ("1");

list 1.addItem ("2");

list 1.addItem ("3");

2. Создать пустой список, добавить пункты при помощи *addItem(),* при этом можно ограничить количество видимых в окне списка пунктов. Ниже по­казан пример, демонстрирующий список, в котором видно 2 элемента.

Пример 3.5

List list2 = new List (2, true);

list2.additem ("1");

list2.additem ("2");

list2.additem ("3");

Некоторые полезные методы класса *List:*

*String getItem(int) -* получить текст пункта с номером *int: int*

*countItems() -* посчитать количество пунктов в списке;

*void clear() -* очистить список;

*void delItem(int) -* удалить из списка пункт с номером *int:*

*int getSelectedIndex() -* получить порядковый номер выделенного элемен­та списка;

*voidselect(int) -* выделить элемент списка с определенным номером.