МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Методические указания для выполнения

лабораторных работ

по дисциплине

Проектирование человеко-машинного интерфейса

для студентов специальности

231000.62

«Программная инженерия»

Томск – 2012

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации

Утверждаю:
Зав. каф. АОИ
профессор
_____Ю.П. Ехлаков
«__» _____2012 г.

Методические указания по выполнению

лабораторных работ по дисциплине

Проектирование человеко-машинного интерфейса

для студентов специальности
231000.62 «Программная инженерия»

Разработчик:
Ст. преподаватель каф. АОИ
_____ Т.А. Петкун
_____ _____2012

Томск-2012

ВВЕДЕНИЕ

Целью выполнения лабораторных работ является получение студентами навыков разработки пользовательского интерфейса, закрепление знаний о требованиях к средствам отображения информации и ввода данных, методах и процедурах разработки и оценки взаимодействия «человек-компьютер».

Процесс выполнения работ направлен на формирование следующих компетенций:

1. общекультурные компетенции - владение культурой мышления, способности к обобщению, анализу, восприятию информации, постановке цели и выбору путей ее достижения (ОК-1).
2. готовность к кооперации с коллегами, работе в коллективе (ОК-3);
3. способность создавать программные интерфейсы (ПК-14);
4. навыки использования операционных систем, сетевых технологий, средств разработки программного интерфейса, применения языков и методов формальных спецификаций, систем управления базами данных (ПК-15);
5. умение применять основные методы и инструменты разработки программного обеспечения (ПК-17).

В результате выполнения лабораторных работ должны быть сформированы умения учитывать человеческий фактор в процессе разработки пользовательского интерфейса, а также наиболее характерные ошибки и пути их предотвращения. Студент должен знать основные факторы, влияющие на качество человеко-машинного взаимодействия; пути и методы оценки и создания качественного пользовательского интерфейса. Владеть навыками проектирования и создания пользовательских интерфейсов; использования наиболее распространенных программно-инструментальных средств создания качественного человеко-компьютерного взаимодействия.

Лабораторная работа №1 Создание интерфейса к задаче вычислительной математики с использованием графики

<u>Цель работы</u>: Получение навыков проектирования интерфейсов для задач вычислительной математики

Теоретический материал для лабораторных работ №1 и №2.

У ряда объектов из библиотеки визуальных компонент есть свойство Canvas (канва), которое предоставляет простой путь для рисования на них. Эти объекты - TBitmap, TComboBox, TDBComboBox, TDBGrid, TDBListBox, TDirectoryListBox, TDrawGrid, TFileListBox, TForm, TImage, TListBox, TOutline, TPaintBox, TPrinter, TStringGrid. Canvas является в свою очередь объектом, объединяющим в себе поле для рисования, карандаш (Pen), кисть (Brush) и шрифт (Font). Canvas обладает также рядом графических методов : Draw, TextOut, Arc, Rectangle и др. Рассмотрим подробнее свойства и методы объекта Canvas.

Использование перьев

С помощью перьев на канве рисуются линии. Доступ к перьям осуществляется через свойство Canvas.Pen Для изменения способа рисования линий следует модифицировать свойства объекта пера: Color, Width, Style, и Mode.

Свойство Color определяет цвет пера. В Delphi предусмотрены предопределенные цветовые константы, например, clRed, clYellow соответствуют красному и желтому цветам. Кроме того, определены константы для представления системных цветов экранных элементов системы Win32. Например, константы clActiveCaption и clHighliteText соответствуют цветам активных заголовков и выделенного текста в данной системе Win32. Приведенная ниже строка назначает перу канвы синий цвет.

```
Canvas.Pen.Color := clBlue;
```

С помощью пера можно рисовать линии, отличающиеся друг от друга стилем, который определяется свойством Style. В таблице показаны различные стили, которые можно устанавливать для свойства Pen.Style:

Стиль	Что рисуется
psClear	Невидимая линия
psDash	Линия, состоящая из штрихов
psDashDot	Линия, состоящая из чередующихся штрихов и точек
psDashDotDot	Линия, состоящая из сочетания штрих-точка-точка
psDot	Линия, состоящая из точек
psInsideFrame	Линия внутри рамки замкнутой формы, определяющей ограничивающий прямоугольник
psSolid	Сплошная линия

Следующая строка демонстрирует, как изменить стиль рисования пером:

```
Canvas.Pen.Stylel := psDashDot;
```

Свойство `Pen.Width` позволяет указать ширину линии в пикселях, проводимую пером при рисовании. При установке большего значения перо рисует более толстую линию.

Использование кисти

Если перо (объект `TPen`) позволяет рисовать на канве линии, то с помощью кисти (объекта `TBrush`) выполняется закрашивание областей и фигур, нарисованных на этой канве. При этом могут использоваться различные цвета, стили и узоры.

Объект `TBrush` обладает тремя важными свойствами `Color, Style` и `BitMap`, которые определяют, как кисть будет закрашивать поверхность канвы. Свойство `Color`

определяет цвет кисти, Style — узор фона кисти, а BitMap задает растр, который можно использовать для создания пользовательских орнаментов, служащих фоном кисти.

Свойство Style кисти может принимать одно из восьми допустимых значений: bsSolid, bsClear, bsHorizontal, bsVertical, bcFDiagonal, bsBDiagonal, bsCross и bsDiagCross. По умолчанию цвет кисти устанавливается равным константе clWhite, стиль — константе bsSolid, а растр не задается. В вашей власти изменять стандартные цвет и стиль работы кисти при выполнении закрашивания областей различными узорами.

Использование шрифтов

С помощью свойства Canvas.Font можно выводить на канву текст, используя любой из доступных в интерфейсе Win32 шрифтов. При этом существует возможность изменять внешний вид помещаемого на канву текста путем модификации такого свойства шрифта, как Color, Name, Size, Height или Style.

Свойству Font.Color можно присвоить любой определенный в Delphi цвет. Например, с помощью следующей строки можно получить шрифт красного цвета:

```
Canvas.Font.Color := clRed;
```

Свойство Name предназначено для указания имени шрифта Windows. Так, используя следующую строку, в качестве шрифта канвы можно установить любой требуемый шрифт:

```
Canvas.Font.Name := 'Times New Roman';
```

Свойство Canvas.Font.Size задает размер шрифта в пунктах.

Свойство Canvas.Font.Stile представляет собой множество, которое может состоять из одного или произвольной комбинации стилей, приведенных в таблице.

Значение	Стиль
fsBold	Полужирный
fsItalic	Курсив
fsUnderline	Подчеркивание
fsStrikeOut	Перечеркнутый текст

Для объединения двух стилей используйте следующий синтаксис:

```
Canvas.Font.Style := [fsBold, fsItalic];
```

Для выбора определенного шрифта и присвоения его свойству Tmemo.Font можно использовать объект TfontDialog:

```
If FontDialog1.Execute then
   Memo1.Font.Assign(FontDialog1.Font);
```

Тот же результат будет достигнут, если присвоить шрифт, выбранный с помощью объекта TfontDialog, свойству объекта Canvas, отвечающему за шрифт:

```
Canvas.Font.Assign(FontDialog1.Font);
```

Кроме того, шрифту объекта Canvas можно присвоить отдельные атрибуты, принадлежащие шрифту, выбранному с помощью объекта TfontDialog:

```
Canvas.Font.Name := Font.Dialog1.Font.Name;
Canvas.Font.Size := Font.Dialog1.Font.Size;
```

В классе Tcanvas предусмотрены различные методы воспроизведения фигур на канве: Arc(), Chord(), LineTo(), Pie(), Polygon(), PolyLine(), Rectangle(), RoundRect(). При прорисовке линий в этих методах используются карандаш (Pen) канвы, а для заполнения внутренних областей - кисть (Brush). Чтобы нарисовать эллипс, используем метод Ellipse(), как показано в следующей строке:

```
Canvas.Ellipse(0,0,ClientWidth, ClientHight);
```

Можно также выполнить заливку любой области канвы с помощью узора кисти, заданного в свойстве Canvas.Brash.Style. При выполнении следующего фрагмента будет

нарисован эллипс, закрашенный в соответствии со значением свойства `Canvas.Brash.Style`:

```
Canvas.Brash.Style := bsCross;
Canvas.Ellipse(0,0, ClientWidth, ClientHight);
```

Полезные графические примитивы для построения графиков: `LineTo(x,y)`– установить карандаш в точке с координатами `(x,y)`; `MoveTo(x,y)`– провести линию от текущей точки к точке с координатами `(x,y)`.

Задание.

Спроектировать и реализовать интерфейс к одному из известных методов задач вычислительной математики. В интерфейсе предусмотреть удобный ввод исходных данных, вывод результата предусмотреть в виде графика. В отдельном диалоговом окне задаются параметры графика (цвет, толщина, стиль линий и т.п.).

Лабораторная работа №2. Реализация интерфейса для визуального решения задачи «Геометрия на плоскости»

<u>Цель работы</u>: получение навыков создания интерфейсов, использующих методы прямого манипулирования

Требования к создаваемому интерфейсу:

Решение задачи оформить в виде проекта на Delphi, в котором следует предусмотреть:

- Специальное окно, содержащее компонент PaintBox для графического представления данных.
- Таблицу StringGrid для табличного представления данных.
- Главное меню со следующими пунктами:
  - Данные → Создать — ввод нового набора данных;
  - Данные → открыть — чтение набора исходных данных из текстового файла (использовать OpenDialog);
  - Расчет — вычисления по условию задания;
  - Результат → Рисунок — представление результата на графике;
  - Результат → Файл — вывод результата расчета в текстовый файл (использовать SaveDialog);
  - Справка → О программе — окно с текстом условия задания.
- Кнопку, запускающую процедуру расчета.
- Рекомендуемый размер графической области 400*400 пикселей для квадрата $(-100 \le x \le 100,\ \ -100 \le y \le 100)$.
- Возможность добавления новых точек щелчком по графику.
- Возможность изменением в таблице.
- Механизм переноса точек по графической области.

Варианты заданий:

1. Дано N точек на плоскости. Выбрать из них три такие, чтобы была наибольшая площадь треугольника с вершинами в этих трех точках.
2. Дано N точек на плоскости. Выбрать из них три такие, чтобы был наибольшим периметр треугольника с вершинами в этих трех точках.
3. Заданы три точки на плоскости. Определить, принадлежит ли начало координат (0, 0) треугольнику, образованному этими точками.

Лабораторная работа №3. Панели инструментов, Компонент TStringGrid

<u>Цель работы</u>: получение навыков создания интерфейсов для задач, требующих визуализации двумерных массивов (таблиц).

Теоретический материал для выполнения работы.

Для визуализации работы с двумерным массивом будем использовать компонент TStringGrid со страницы Additional палитры компонентов, предназначенный для создания таблиц, в ячейках которых располагаются произвольные текстовые строки

Таблица делится на две части — фиксированную и рабочую. Фиксированная служит для показа заголовков столбцов/рядов и для ручного управления их размерами. Обычно фиксированная часть занимает крайний левый столбец и самый верхний ряд таблицы, однако с помощью свойств `FixedCols` и `FixedRows` можно задать другое количество фиксированных столбцов и рядов (если эти свойства имеют значение 0, таблица не содержит фиксированной зоны). Рабочая часть — это остальная часть таблицы. Она может содержать произвольное количество столбцов и рядов, более того, эти величины могут изменяться программно. Рабочая часть может не умещаться целиком в пределах окна компонента, в этом случае в него автоматически помещаются нужные полосы прокрутки. При прокрутке рабочей области фиксированная область не исчезает, но меняется ее содержимое — заголовки строк и рядов.

Свойство	**Тип**	**Комментарий**
`Cells[ACol, ARow: Integer]`	`String`	Определяет содержимое ячейки с табличными координатами (`ACol, ARow`)
`Cols[ACol: Integer]`	`TStrings`	Все строки с номером `ACol`
`Rows[ACol: Integer]`	`TStrings`	Все строки с номером `ARCow`
`ColCount`	`Integer`	Общее количество столбцов
`RowCount`	`Integer`	Общее количество рядов
`FixedCols`	`Integer`	Количество фиксированных (заголовочных) столбцов
`FixedRow s`	`Integer`	Количество фиксированных (заголовочных) рядов
`Col`	`Integer`	Номер столбца текущей ячейки
`Row`	`Integer`	Номер ряда текущей ячейки
`Options`	`TGridOptions`	Данное свойство множественного типа определяет ряд дополнительных параметров таблицы. В частности, нам наиболее важны следующие:
	`goEditing`	Разрешается редактирование ячеек таблицы
	`goTabs`	Разрешается перемещение по ячейкам с помощью клавиши `Tab` и комбинации `Shift+Tab`

Центральным свойством компонента является `Cells` — двумерный массив ячеек, каждая из которых может содержать произвольный текст. Конкретная ячейка определяется парой чисел — номером столбца и номером ряда, на пересечении которых она находится

(нумерация начинается с нуля). Свойство Cells имеет тип String, поэтому программа может легко прочитать или записать содержимое нужной ячейки. Например:

```
Cells[1,1] := 'Верхняя ячейка рабочей зоны';
```

Количество ячеек по каждому измерению хранит пара свойств ColCount (количество столбцов)и RowCount (количество рядов). Значения этих свойств, и, следовательно, размеры таблицы могут меняться как на этапе разработки программы, так и в ходе ее работы, однако их значения должны быть как минимум на единицу больше соответственно значений в свойствах FixedCols и FixedRows, определяющих размеры фиксированной зоны. Содержимое ячеек можно редактировать.

В данной работе наше приложение должно быть красиво оформлено в виде пиктограмм. Пиктограммы будут назначаться пунктам меню и кнопкам панели инструментов. Для того, чтобы можно было назначить картинку любому из этих элементов, необходимо вначале установить в свойстве ImageList содержащего его компонента используемый список картинок, а затем в свойстве ImageIndex каждого элемента указать номер картинки из списка. Однако если наша программа основана на концепции действий (Action), вместо прямого задания значения свойства ImageIndex пункта меню или кнопки панели инструментов следует указать значение свойства ImageIndex соответствующего компонента TAction.

Для определения действия необходимо на форму поместить компонент типа TActionList, который может содержать в себе множество действий. Затем нужно дважды щелкнуть на нем мышкой, при этом откроется редактор действий. Нажатием на клавишу Insert можно добавлять новые действия. При выборе мышкой действия в списке, оно становится доступным в инспекторе объектов. Список основных свойств приведен ниже в таблицах 1-2.

Таблица 1.Основные свойства объектов типа TAction

Свойство	**Тип**	**Комментарий**
Caption	**String**	Название действия в меню
Category	String	Категория — используется для упорядочивания действий внутри TActionList
Checked	Boolean	Отмечены ли галочкой пункты меню и нажаты ли соответствующие кнопки
Enabled	Boolean	Разрешена ли команда
Hint	String	Всплывающая подсказка для кнопок
ImageIndex	Integer	Код картинки в связанном списке картинок
ShortCut	TshortCut	Код горячей клавиши для вызова действия
Visible	Boolean	Видимы ли пункты меню и кнопки

Таблица 2. Основные события объектов типа TAction

Свойство	**Комментарий**
OnExecute	Выполнение действия
OnUpdate	Обновление информации о действии. Здесь можно изменить любые свойства действия в зависимости от текущего состояния программы. Обычно с помощью этого события запрещают недоступные команды.

После того, как созданы все необходимые действия, их необходимо назначить соответствующим пунктам меню и кнопкам. У этих компонентов имеется свойство Action, значение которого можно установить в инспекторе объектов с помощью выпадающего списка всех доступных в форме действий.

В нашем приложении необходимо сделать назначение действий для главного меню, локального меню и кнопкам панели.

При создании элементов меню, соответствующих действиям, не нужно указывать заголовки. Нужно задать свойство `Action` элемента меню, при этом заголовок, код картинки, горячая клавиша и обработчик события будут автоматически взяты из компонента `TAction` .

Замечание *Не забудьте указать свойство `ImageList` компонетов меню и панели инструментов из выпадающего списка в инспекторе объектов для отображения в них картинок.*

Локальное меню обычно появляется на экране при нажатии правой кнопки мыши на визуальных компонентах. Для этого у этих компонентов должно быть установлено свойство `PopupMenu` в инспекторе объектов. В нашем случае достаточно назначить свойство `PopupMenu` только для формы. Тогда нажатие правой кнопки мышки в любом месте формы будет вызывать локальное меню.

Для создания панели инструментов с кнопками используем компонент `Ttoolbar`. Для добавления в него новых кнопок необходимо нажать правую кнопку мыши для вызова локального меню и выбрать соответствующий пункт. Как и пункты меню, кнопки имеют свойство `Action`, при установке значения которого из выпадающего списка кнопки автоматически получат все необходимые свойства (текст надписи, всплывающую подсказку, код картинки, обработчик нажатия). По умолчанию кнопки на панели инструментов отображаются только с картинкой без сопроводительной надписи. Для отображения с надписью необходимо указать свойство `ShowCaption` равным `True`.

В большинстве современных приложений панели инструментов можно свободно перетаскивать по экрану с помощью мышки.

Для того, чтобы это можно было делать, в Delphi имеется компонент типа `TControlBar`, который дополняет все помещаемые на него другие компоненты рамочкой и двумя вертикальными полосками слева для перетаскивания. Обычно в `TControlBar` помещают панели инструментов `TToolBar`. Поэтому перед размещением панелей инструментов на форме сначала необходимо поместить компонент `TControlBar`, а в него затем — панели инструментов.

Задание

Создать проект, реализующий работу с двумерным массивом. Приложение должно быть снабжено главным меню, локальным меню и панелью инструментов.

Таблица 3. Список действий, создаваемых в приложении

Действие	**Описание**
ActionInput	Заполнить матрицу с помощью датчика случайных чисел
ActionClear	Очистить матрицу. Вернуть первоначальную размерность
`ActionMatr`	Смотрите задание
`ActionExit`	Выход из программы
`ActionAbout`	Выдает краткую информацию об авторе

1. Удалить строку, содержащую максимальный элемент.
2. Удалить столбец, содержащий максимальный элемент.
3. Отсортировать матрицу по возрастанию элементов первого столбца.
4. Отсортировать матрицу по убыванию элементов первой строки.

5. Удалить строку и столбец, на пересечении которых находится минимальный элемент матрицы.

Лабораторная работа №4. Создание MDI-приложения

<u>Цель работы</u>: получение навыков создания многодокументных интерфейсов.

Теоретический материал для выполнения работы.

Термин MDI (Multiple Document Interface) дословно означает многодокументный интерфейс и описывает приложения, способные загрузить и использовать одновременно несколько документов или объектов. Примером такого приложения может служить диспетчер файлов (FileManager).

Обычно MDI-приложения состоят минимум из двух форм — родительской и дочерней. Свойство родительской формы FormStyle установлено равным fsMDIForm. Для дочерней формы установите стиль fsMDIChild.

Родительская форма служит контейнером, содержащим дочерние формы, которые заключены в клиентскую область и могут перемещаться, изменять размеры, минимизироваться или максимизироваться. В вашем приложении могут быть дочерние формы разных типов, например одна — для обработки изображений, а другая — для работы с текстом.

Создание форм

В MDI-приложении, как правило, требуется выводить несколько экземпляров классов формы. Поскольку каждая форма представляет собой объект, она должна быть создана перед использованием и освобождена, когда в ней больше не нуждаются. Delphi может делать это автоматически, а может предоставить эту работу вам.

Автоматическое создание форм

По умолчанию при запуске приложения Delphi автоматически создает по одному экземпляру каждого класса форм в проекте и освобождает их при завершении программы. Автоматическое создание обрабатывается генерируемым Delphi кодом в трех местах.

Первое — раздел интерфейса в файле модуля формы.

```
type
  TForm1 = class (TForm)
  private
  {Закрытые объявления.}
  public
  {Открытые объявления.}
  end;
```

В данном фрагменте кода объявляется класс TForm1.
Вторым является место, в котором описывается переменная класса.

```
var Form1: TForm1;
```

Здесь описана переменная Form1, указывающая на экземпляр класса TForm1 и доступная из любого модуля. Обычно она используется во время работы программы для управления формой. Третье место находится в исходном тексте проекта, доступ к которому можно получить с помощью меню View/ Project Source. Этот код выглядит как:

Application.CreateForm(TForm1, Form1);

Процесс удаления форм обрабатывается с помощью концепции владельцев объектов: когда объект уничтожается, автоматически уничтожаются все объекты, которыми он владеет. Созданная описанным образом форма принадлежит объекту Application и уничтожается при закрытии приложения.

Динамическое создание форм

Хотя автоматическое создание форм полезно при разработке SDI-приложений, при создании MDI-приложении оно, как правило, неприемлемо.
Для создания нового экземпляра формы используйте конструктор Create класса формы. Приведенный ниже код создает новый экземпляр TForm1 во время работы программы и устанавливает его свойство Caption равным 'New Form'.

```
Form1:= TForm1.Create(Application);
Form1.Caption:= 'New Form';
```

В приведенном ниже коде Form1 указывает только на последнюю созданную форму. Если вам это не нравится, воспользуйтесь приведенным ниже кодом — возможно, он более точно отвечает вашим запросам:

```
with TFormI.Create(Application) do
  Caption:= 'New Form';
```

Замечание: При разработке MDI-приложения метод Show не нужен, так как Delphi автоматически показывает все вновь созданные дочерние MDI-формы. В случае SDI-приложения вы обязаны использовать метод Show.

MDI-свойства TForm

Объект TForm имеет несколько свойств, специфичных для MDI-приложений.

ActiveMDIChild

Это свойство возвращает дочерний объект TForm, имеющий в текущее время фокус ввода. Оно полезно, когда родительская форма содержит панель инструментов или меню, команды которых распространяются на открытую дочернюю форму.

Например, представим, что проект использует дочернюю форму, содержащую элемент TMemo, названный memDailyNotes. Имя класса этой дочерней формы— TfrmMDIChild. Родительская форма содержит кнопку Clear в панели инструментов, которая удаляет содержимое memDailyNotes в активной дочерней форме. Вот как это реализуется.

```
procedure TfrmMDIParent.spbtnClearClick(Sender: TObject);
begin
  if not (ActiveMDIChild = Nil) then
    if ActiveMDIChild is TfrmMDIChild then
      TfrmMDIChild(ActiveMDIChild).memDailyNotes.Clear;
end;
```

В первой строке проверяется, равен ли ActiveMDIChild значению Nil, так как в этом случае обращение к объекту вызовет исключительную ситуацию.

Поскольку ActiveMDIChild возвращает объект TForm, компилятор не имеет доступа к memDailyNotes — объекту TfrmMDIChild. Вторая строка проверят соответствие типов, т.е. действительно ли ActiveMDIChild указывает на объект TfrmMDIChild.

Третья строка выполняет преобразование типа и вызывает метод Clear компонента memDailyNotes.

MDIChildren и MDIChildCount

Свойство MDIChildren является массивом объектов TForm, предоставляющих доступ к созданным дочерним формам. MDIChildCount возвращает количество элементов в массиве MDIChildren.

Обычно это свойство используется при выполнении какого-либо действия над всеми открытыми дочерними формами. Вот код сворачивания всех дочерних форм командой Minimize All.

```
procedure TFormI.mnuMinimizeAllClick(Sender: TObject);
var
   iCount: Integers;
begin
   for iCount:= MDIChildCount-1 downto 0 do
      MDIChildren[iCount].WindowState:= wsMinimized;
end;
```

Если вы будете сворачивать окна в порядке возрастания элементов массива, цикл будет работать некорректно, так как после сворачивания каждого окна массив MDIChildren обновляется и пересортировывается, и вы можете пропустить некоторые элементы.

TileMode

Это — свойство перечислимого типа, определяющее, как родительская форма размещает дочерние при вызове метода Tile. Используются значения tbHorizontal (по умолчанию) и tbVertical для размещения форм по горизонтали и вертикали.

WindowMenu

Профессиональные MDI-приложения позволяют активизировать необходимое дочернее окно, выбрав его из списка в меню. Свойство WindowMenu определяет объект TMenuItem, который Delphi будет использовать для вывода списка доступных дочерних форм.

Для вывода списка TMenuItem должно быть меню верхнего уровня. Это меню имеет свойство Caption, равное swindow.

MDI-события TForm

В MDI-приложении событие OnActivate запускается только при переключении между дочерними формами. Если фокус ввода передается из не MDI-формы в MDI-форму, генерируется событие OnActivate родительской формы, хотя ее свойство Active никогда и не устанавливается равным True. Эта странность на самом деле строго логична: ведь, если бы OnActivate генерировался только для дочерних форм, не было бы никакой возможности узнать о переходе фокуса ввода от другого приложения.

MDI-методы TForm

Специфичные для MDI-форм методы перечислены ниже. Arrangelcons выстраивает пиктограммы минимизированных дочерних форм в нижней части родительской формы.

Cascade располагает дочерние формы каскадом, так что видны все их заголовки. Next и Previous переходит от одной дочерней формы к другой, как будто вы нажали <Ctrl+Tab> или <Ctrl+Shift+Tab>.

Tile выстраивает дочерние формы так, что они не перекрываются.

Работа с меню

Существуют определенные особенности работы меню в MDI-приложениях. Покажем, каким образом, MDI-приложение позволяет своим дочерним формам совместно использовать одну и ту же строку меню с помощью технологии *слияния меню*.

Если, к примеру, в строке меню дочерней формы TmdiEditFrm находятся два отдельных меню — File, Edit, Character. Строка меню родительской формы TmainForm содержит два элемента —File и Window. При активизации дочернего окна видно меню из четырех элементов — File, Edit, Character и Window. Меню File родительской формы и меню File дочерней формы содержат различные команды. Для того, чтобы получить этот эффект, следует использовать свойство GroupIndex. В нашем случае свойство GroupIndex меню File должно иметь значение 0, а это же свойство меню Edit, Caracter — значения1. Как и в случае дочерней формы, свойство GroupIndex меню File родительской формы имеет значение 0. Однако значение свойства GroupIndex меню Window — 9.

Свойство GroupIndex играет большую роль, так как именно оно управляет процессом слияния меню. Это означает, что, когда главная форма запускает дочернюю форму, меню последней сливается с меню главной формы. Свойство GroupIndex определяет, в каком порядке следуют отдельные меню и какие меню главной формы заменяются меню дочерней формы. Слияние применяется лишь к элементам строки меню компонента TmainMenu, а не к их командам.

Если значение свойства GroupIndex элемента меню дочерней формы совпадает со значением свойства GroupIndex элемента меню главной формы, элемент меню дочерней формы заменяет собой элемент меню главной формы. Оставшиеся меню располагаются в строке в порядке, определенном значениями свойств GroupIndex элементов объединенного меню.

Слияние меню в MDI-приложениях выполняется автоматически. Если значения свойств GroupIndex элементов меню установлены в требуемом порядке, всегда будет присходить корректное слияние элементов мню при вызове дочерних MDI-форм.

Добавление в меню списка открытых документов

Для добавления в меню Window списка открытых документов необходимо присвоить свойству WindowMenu главной формы экземпляр элемента меню, содержащего требуемый список открытых документов. Например, свойству TmainForm.WindowMenu присваивается объект mmiWindow, ссылающийся на меню Window в строке меню приложения.

Для того, чтобы обеспечить отображение панели инструментов в родительском окне, нужно написать для дочерних форм следующие обработчики.

```
procedure TMDIChildForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
  {Переопределение объекта parent панели инструментов }
  tlbMain.Parent := self;
  { Если это была последняя дочерняя форма, то делаем видимой панель инструментов главной
формы }
  if (MainForm.MDIChildCount = 1) then
    MainForm.tlbMain.Visible := True
end;
```

В обработчике события TMDIChildForn.FormClose() параметру Action назначается значение caFree с целью гарантированного уничтожения экземпляра формы TMDIChildForn при ее закрытии. Переменная Action перечислимого типа может получить одно из 4 допустимых значений:

- caNone. Ничего не выполняется.
- caHide. Форма удаляется с экрана, но не уничтожается.

- caFree. Форма освобождается.
- caMinimize. Форма минимизируется (выполняется по учолчанию).

```
procedure TMDIChildForm.FormActivate(Sender: TObject);
begin
  { Когда форма становится активной, панель инструментов главной формы следует убрать,
после чего назначить и вывести в родительской форме панель инструментов данного дочернего
окна }
  MainForm.tlbMain.Visible := False;
  tlbMain.Parent := MainForm;
  tlbMain.Visible := True;
end;

procedure TMDIChildForm.FormDeactivate(Sender: TObject);
begin
  { Дочерняя форма становится неактивной либо при ее уничтожении, либо в том случае, когда
активной становится другая дочерняя форма. Убираем панель инструментов этой формы, чтобы
можно было сделать видимой панель инструментов другой формы }
  tlbMain.Visible := False;
end;
```

Когда дочерняя форма становится активной, вызывается ее обработчик события OnActivate(). Всякий раз, когда дочерняя форма становится активной, выполняются определенные действия. Прежде всего, делается невидимой панель инструментов главной формы, что позволяет сделать видимой панель инструментов дочерней формы. Кроме того, главная форма назначается родительским объектом панели инструментов дочернего окна — поэтому данная панель инструментов будет отображаться в окне главной формы, а не дочерней. В обработчике события OnDeactivate() панель инструментов дочерней формы делается просто невидимой. Наконец, при обработке события OnClose() дочерняя форма вновь назначается в качестве родительского объекта ее панели инструментов, и если текущая дочерняя форма являлась единственной в приложении, то делается видимой панель инструментов главного окна. В результате при работе приложения создается впечатление, что главная форма содержит единственную панель инструментов, состав кнопок которой изменяется в соответствии с типом активного дочернего окна.

Задание.

Создать MDI-приложение, реализующее просмотрщик текстовых файлов и картинок. Приложение должно быть снабжено меню, панелями инструментов, при этом панель инструментов дочернего окна должна отображаться в родительском окне.

Лабораторная работа № 5. Создание собственных компонент

Цель работы: получение навыков создания новых компонентов, отвечающих требованиям пользователя.

Теоретический материал для выполнения работы.

Поскольку Delphi является открытой средой и позволяет не только использовать объекты из Библиотеки Визуальных Компонент (VCL) в своей программе, но и создавать новые объекты. Причем, ничего другого, кроме Delphi, для этого не требуется. Создание нового объекта в Delphi не является очень сложной задачей, хотя для этого и требуется знание Windows API, объектно-ориентированного программирования и иерархии классов в VCL.

Может возникнуть вопрос; если в Delphi уже есть своя библиотека, то зачем еще создавать какие-то объекты? Ответ прост: нельзя создать библиотеку на все случаи жизни и на все вкусы. Новые компоненты, во-первых, позволяют расширить область применения Delphi: например, с помощью библиотек объектов третьих фирм разрабатывать приложения для работы в Internet. Во-вторых, позволяют дополнить или настроить для себя имеющиеся в VCL объекты (например, переопределить значения свойств, устанавливаемые по умолчанию).

Соглашения по наименованиям

Если вы рассматривали исходные тексты VCL, то могли видеть, что они следуют нескольким простым соглашениям при определении новых классов. Delphi этого не требует, имена методов, свойств и т.п. могут быть любыми, компилятору это безразлично. Но если следовать этим соглашениям, то разработка новых компонентов и чтение исходных текстов станет существенно проще.

Итак:

- Все декларации типов начинаются на букву T. Еще раз, Delphi не требует этого, но это делает очевидным, что "TEdit", например, есть определение типа, а не переменная или поле класса.
- Имена свойствам нужно давать легко читаемые и информативные. Нужно помнить, что пользователь будет их видеть в Инспекторе Объектов. И имя вроде "TextOrientation" много удобнее, нежели "TxtOr". То же самое относится к методам. Методы, доступные пользователю, должны иметь удобные названия.
- При создании свойств типа Event, имя такого свойства должно начинаться с “On” (например, OnClick, OnCreate и т.д.).
- Имя метода для чтения свойства должен начинаться со слова “Get”. Например, метод GetStyle должен выполнять чтение для свойства Style.
- Имя метода для записи свойства должен начинаться со слова “Set”. Например, метод SetStyle должен выполнять запись в свойство Style.
- Внутреннее поле для хранения данных свойства должно носить имя, начинающееся с буквы “F”. Например, свойство Handle могло бы храниться в поле FHandle.

Конечно же, есть исключения из правил. Иногда бывает удобнее их нарушить, например, класс TTable имеет свойства типа Event, которые называются BeforePost, AfterPost и т.п.

Выбор предка

Прежде, чем приступить к написанию кода, нужно определиться, хотя бы приблизительно, что за компонент вы собираетесь делать. Далее, исходя из его предполагаемых свойств, определите класс-предок. В VCL имеется несколько базовых классов, рекомендуемых для наследования:

- **TObject** - Можно использовать в качестве предка, если с этим компонентом не нужно работать во время дизайна. Это может быть, например, класс, содержащий значения переменных среды (environment) или класс для работы с INI файлами.
- **TComponent -** Отправная точка для многих невидимых компонент. Данный класс обладает встроенной возможностью сохранять/считывать себя в потоке во время дизайна.
- **TGraphicControl** - Используйте этот класс для создания видимых компонент, которым не нужен handle. Такие компоненты рисуют прямо на своей поверхности и требуют мало ресурсов Windows.
- **TWinControl** - Базовый класс для компонент, которые имеют окно. Данное окно имеет свой handle, его используют при доступе к возможностям Windows через API.
- **TCustomControl** - Потомок TWinControl, вводит понятие канвы (Canvas) и метод Paint() для лучшего контроля за прорисовкой компонента. Именно этот класс используется в качестве базового для построения большинства видимых компонент, имеющих оконный handle.
- **TXxxxx** - Класс вроде TEdit или TButton. Используются с целью доопределения их свойств и методов или переопределения значения свойств, принимаемых по умолчанию.

Пример создания компонента

Для примера создадим новый класс, мутант TButton, в котором изменим значение по умолчанию свойства ShowHint на True и добавим новое свойство - счетчик нажатий на кнопку. Заготовка модуля для создания нового компонента уже есть (см. пункт **Заготовка для нового компонента**). Теперь исходный текст выглядит так:

```
unit New_btn;
interface
uses
SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls, Forms, Dialogs, StdCtrls;
type
TMyButton = class(TButton)
private
{ Private declarations }
FClickCount : Longint;
protected
{ Protected declarations }
public
{ Public declarations }
constructor Create(AOwner : TComponent); override;
procedure Click; override;
property ClickCount : Longint read FClickCount write
FClickCount;
published
{ Published declarations }
end;
procedure Register;
implementation
constructor TMyButton.Create(AOwner : TComponent);
begin
inherited Create(AOwner);
ShowHint:=True;
FClickCount:=0;
end;
procedure TMyButton.Click;
begin
Inc(FClickCount);
inherited Click;
end;
procedure Register;
begin
RegisterComponents('Samples', [TMyButton]);
end;
end.
```

Для того, чтобы переопределить начальное значение свойства при создании объекта, нужно переписать конструктор *Create,* в котором и присвоить этому свойству нужное значение (не забыв перед этим вызвать конструктор предка).

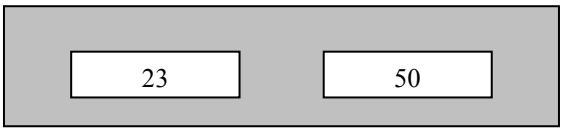
Новое свойство для подсчета нажатий на клавишу называется *ClickCount.* Его внутреннее поле для сохранения значения - FClickCount имеет тип Longint, емкости поля хватит надолго.

Задания к самостоятельной работе

1. Создать компонент, позволяющий в строке ввода задавать целые числа. В компоненте предусмотреть возможность задания допустимого диапазона значений, количества позиций, выравнивания внутри строки ввода.

2. Создать компонент, позволяющий в строке ввода задавать вещественные числа. В компоненте предусмотреть возможность задания допустимого диапазона значений, количество позиций на целую и дробную части.

3. Создать компонент, позволяющий вводить данные в следующем виде:



Второе число должно быть всегда больше первого!

4. Создать компонент, который бы позволял перевести курсор в место начала подстроки, которая задается в программе, использующей новый компонент. Если такой фрагмент отсутствует, курсор перевести в конец текста.

5. Написать компонент, который предоставлял бы пользователю следующую возможность: при двойном щелчке мыши над панелью панель разворачивается на 90°.

6. Написать компонент, который бы позволял ввести вещественное число, а при выходе из поля редактирования в нем отображалось (или $\sin(x)$ или $\sqrt{x}$ или x^2).

7. Редактор текста: выполнить компонент, который бы предоставлял следующую возможность: по желанию пользователя автоматически каждое слово начинать с большой буквы, а все другие буквы слова должны быть маленькими.

Лабораторная работа № 6. Создание базы данных. Отчеты.

<u>Цель работы</u>: получение навыков создания программных продуктов, требующих хранения и обработки больших объемов информации.

Теоретический материал для выполнения работы.

Данная лабораторная работа предназначена для знакомства с основами работы с реляционными базами данных. Условно можно считать, что реляционная база данных состоит из набора таблиц, которые в свою очередь состоят из однотипных записей с несколькими полями. Обычно записи в таблицах представляют в виде горизонтальных рядов, а в колонках - значения соответствующих полей.

Базы данных бывают серверного типа, когда они хранятся в специальном формате, и обращение к ним производится только через специальные программы-серверы баз данных, а также бывают локальные базы. В локальных базах обычно отдельные таблицы хранятся в виде одного или нескольких файлов, при этом базой данных считается каталог на диске, содержащий все эти файлы. В нашей работе мы будем работать с локальной базой данных в формате Paradox 7.

В данной работе будет необходимо создать простое приложение для ведения учета книг в домашней библиотеке. Для каждой книги в библиотеке должна быть создана запись в базе данных со следующими полями: фамилия автора, название книги, ее тип и фамилия того, кто взял книгу. Тип книг и список читателей необходимо оформить в виде таблиц-справочников, которые можно будет наполнять по мере необходимости. Просмотр списка книг должен производиться в одном из трех режимов: просмотреть все книги, книги заданного типа, либо книги, находящиеся у выбранного читателя.

В рамках работы необходимо будет освоить программу Database Desktop, предназначенную для создания и редактирования таблиц баз данных. Используя эту программу, нужно будет создать 3 таблицы, которые затем будут подключены к нашей

программе в среде Delphi.

Описание работы

Разработка любой базы данных начинается с проектирования её' логической и физической структур. Физическая схема нашей базы изображена на рис. 24. На ней слева представлена главная таблица, имеющая 4 поля: имя автора, название книги, код типа книги и код читателя

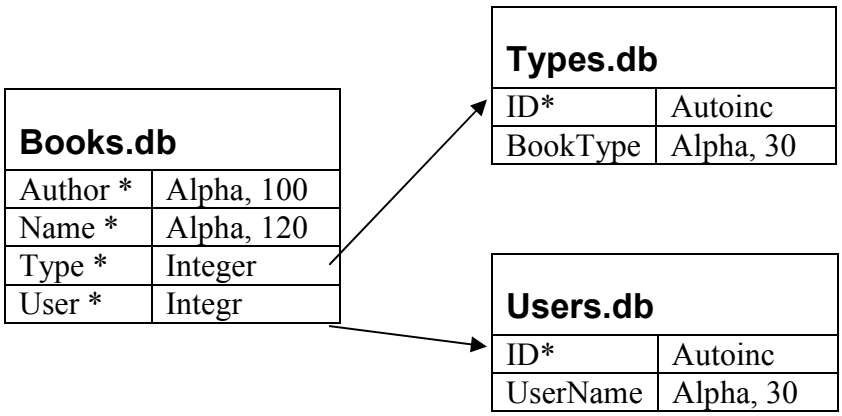


Рис. 6.1. Физическая схема БД домашней библиотеки

Справа представлены две таблицы-справочника, в которых хранится соответствие кода типа книги с текстовым описанием типа и кода читателя с его текстовым именем. Для быстрого поиска и сортировки содержимого таблиц они должны быть проиндексированы по полям, отмеченным звездочками. В первой таблице по полям Author и Name должен быть создан первичный индекс, а по полям Type и User два вторичных. В таблицах-справочниках индекс должен быть создан по полям ID. Он будет использоваться для быстрого поиска текстового описания по коду.

В таблицах-справочниках для полей ID необходимо указать тип данных Autoincrement, что означает, что при добавлении новой записи в таблицу для значения этого поля будет автоматически браться уникальное целочисленное значение. Тип данных Alpha означает текстовую строк)'. Этот . тип имеет целочисленный параметр, означающий максимальную длину строки, которую можно поместить в это поле.

Для физического создания таблиц нашей базы данных можно воспользоваться программой Database Desktop, входящей в состав Delphi. Её можно запустить из Delphi с помощью команды меню Tools {Database Desktop. После ее запуска необходимо выбрать команду меню Filej Working Directory... и затем в диалоге указать каталог, в котором мы будем сохранять таблицы. Пусть это будет тот же каталог, в котором мы сохраняем файлы проекта Delphi.

Затем с помощью команды меню FiJe|New|Table... необходимо создать 3 таблицы, указав список полей таблиц, их типы, а также создать индексы. При выборе этой команды, в первую очередь, в диалоговом окне (рис. 25) необходимо выбрать тип таблицы. Для нашей задачи лучше всего выбрать тип «Paradox 7», т.к. он обладает максимальными возможностями среди других доступных форматов не серверного типа.

После выбора типа таблицы откроется окно создания таблицы формата «Paradox 7», в котором в первую очередь необходимо сформировать список всех полей таблицы. На рис. 25. приведен внешний вид программы Database Desktop в процессе создания новой таблицы, которую мы потом должны сохранить под именем Books.db. В первой колонке списка полей указывается имя поля, во второй — его тип. Для выбора нужного типа можно либо нажать клавишу с первой буквой нужного типа, либо нажать пробел и на экране появится полный список всех доступных типов. В нашей работе нам понадобятся только типы Alpha для представления строковых значений, Long Integer для представления целочисленных кодов типа книги и кода читателя, а также тип + (Autoincrement).

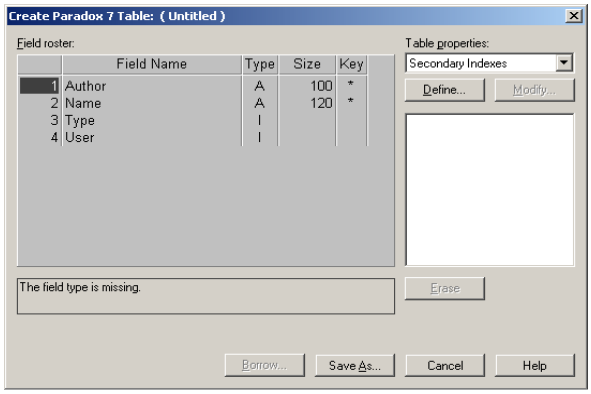


Рис. 6.2. Создание новой таблицы в Database Desktop

Третья колонка списка полей предназначена для задания размера поля. В нашем случае она имеет смысл только для строковых полей. Размер поля определяет максимальную длину строки, которую можно поместить в данное поле.

Четвертая колонка позволяет указать поля, которые будут входить в первичный индекс. Для его задания необходимо просто нажать пробел. Индекс (ключ) используется при отображении данных в таблице в качестве критерия сортировки записей.

В нашей задаче для таблицы . Books.db понадобятся также еще 2 вторичных индекса, в которые будет входить только одно поле, - Type и User соответственно. Для этого в правой верхней части окна создания таблицы необходимо выбрать в выпадающем списке пункт Secondary Indexes. Затем необходимо нажать . на появившуюся кнопку Define, после чего на экране появится диалог создания вторичного индекса — (рис. 6.3).

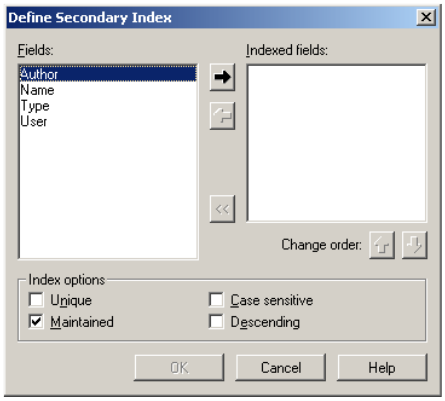


Рис. 6.3. Создание индекса в Database Desktop

В диалоге создания вторичного индекса необходимо выбрать в левом списке имя поля, по которому необходимо создать индекс, и нажать кнопку ⇒. При этом в правом списке отображается список полей, которые будут участвовать во вторичном индексе. В нашем случае необходимо выбрать для индекса поле Type и нажать клавишу OK. При этом появится окно, в котором необходимо указать имя созданного индекса. Обычно для имени индекса указывают имя поля, составляющего индекс, с добавлением слова Index. В нашем случае укажем имя TypeIndex.

После этого необходимо создать еще один вторичный индекс, но уже по полю User. Аналогично сохраним этот индекс под именем UserIndex.

В дальнейшем для изменения структуры уже созданной таблицы в программе Database Desktop необходимо будет открыть её с помощью команды меню File|Open]Table, а затем выбрать команду меню Ta-ble|Restructure...

После физического создания таблиц базы данных можно переходить к созданию пользовательского интерфейса к базе данных в среде Delphi.

В Delphi для подключения к таблице базы данных используется невизуальный компонент типа TTable, находящийся на закладке «Database Access» палитры компонентов. В

простейшем случае для обращения к реальной таблице необходимо указать только 3 свойства, приведенные в табл. 1.

Таблица 1. Основные свойства компонентов типа TTable

Свойство	Тип	Комментарий
DatabaseName	String	Имя базы данных, указанной в компоненте Database, зарегистри-рованном на компьютере в программе BDE Administrator или каталог, содержащий таблицы форматов DBase, FoxPro или Paradox
TableName	String	Имя таблицы указанной базы данных
Active	Boolean	Открыта ли таблица. Для открытия можно также воспользоваться методом Open, а для закрытия- Close

Для удобства разработки компоненты баз данных обычно размещают в отдельных модулях (чтобы компоненты для доступа к базам не путались с остальными): это может быть отдельная пустая невидимая форма или специальный модуль данных (Data Module), который можно создать с помощью пункта меню File! New... В этом модуле необходимо разместить компоненты типа TTable, соответствующие таблицам нашей базы данных.

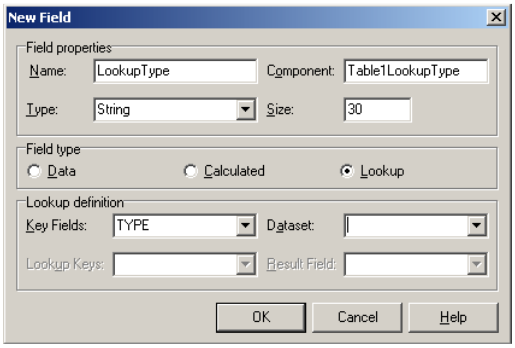


Рис. 6.4. Создание подстановочного поля

Для каждого компонента TTable определен так называемый «курсор», определяющий текущую запись в таблице. Отметим особую роль таблиц-справочников в нашем приложении. Их необходимо будет редактировать при появлении новых читателей и типов книг, а также независимо перемещаться по ним для подмены кодов полей их строковыми значениями. Поэтому для этих таблиц необходимо будет поместить в модуль данных по два компонента TTable, каждый из которых имеет независимые курсоры.

Список полей таблицы представляется в свойстве таблицы Fields. Для его изменения необходимо вызвать редактор списка полей таблицы, который появляется при двойном щелчке мышкой на компоненте TTable. По умолчании список полей пуст, что означает, что он будет формироваться автоматически при открытии таблицы базы данных. Если же мы хотим каким-то образом переопределить свойства полей, то необходимо в локальном меню редактора списка полей вызвать вначале команду Add all fields для получения полного списка полей. Для создания в главной таблице подстановочных полей необходимо выбрать команду локального меню New field...

В главной таблице TableBooks для подстановки текстовых значений вместо кодов типов книг и кодов читателей будем использовать компоненты с именами TableTypesLookup и TableUsersLookup. Поэтому в главной таблице кроме реальных физических полей необходимо создать два дополнительных подстановочных (lookup) поля строкового типа: поля же с кодами необходимо скрыть от пользователя.

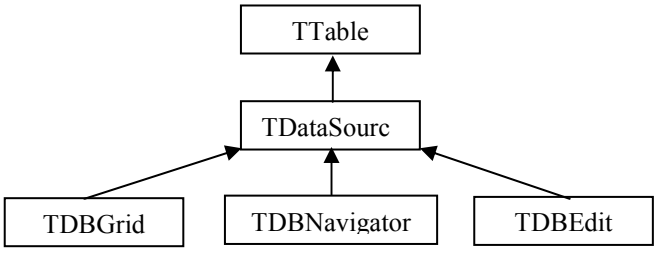
На рис. 6.4 приведен пример создания подстановочного поля типа книги LookupType,

подменяющего числовой код типа книги его текстовым описанием с помощью другой таблицы-справочника TableTypesLookup.

Таблица 2. Основные свойства компонентов типа T Field

Свойство	Тип	Комментарий
Alignment	TAligranent	Выравнивание текста поля при просмотре: по левой стороне, по правой или по центру
DisplayLabel	String	Текст метки поля, используемый, например, в качестве заголовка столбца таблицы
DisplayWidth	Integer	Ширина колонки таблицы по умолчанию в символах
FieldKind	TFieldKind	Определяет, является ли поле отражением реальных данных в физической таблице (f kData), является ли поле подстановочным (f kLookup), вычисляемым (fkCalculated) и т.д.
FieldName	String	Физическое имя поля для типа fkData, либо логическое для других типов
Readonly	Boolean	Разрешено ли изменение поля пользователем
Required	Boolean	Обязательно ли заполнять значение поля при модификации записи
Visible	Boolean	Видно ли поле в таблице по умолчании
LookupDataset	TDataset	Указывает на набор данных (например на таблицу), являющийся справочником
KeyFields	String	Имя поля таблицы, где хранится код, который надо подменить по справочнику
LookupKeyFields	String	Имя поля справочника с кодом
LookupResultField	Spring	Имя поля справочника, возвращаемого как результат подмены кода

Создаваемые поля компонента TTable являются потомками класса TField. Любое поле в списке можно выбрать, и затем изменить его настройки с помощью инспектора объектов. Список основных свойств объектов типа TField приведен в табл. 2.



Связывание компонент для работы с базами данных

После того как таблицы открыты, к ним можно подключать визуальные компоненты для отображения содержимого и перемещения по записям. Для этого в Delphi используется двухступенчатая схема, по которой к таблицам присоединяется один компонент типа

TDataSource, а к нему, в свою очередь, все необходимые визуальные компоненты. В Delphi, как правило, имена типов визуальных компонентов для работы с базами данных начинаются с букв TDB, например, BGrid для отображения всей таблицы в виде сетки, TDBNavigator для навигации по таблице и т.д. Условно иерархия связывания визуальных и невизуальных компонентов для работы с базами данных приведена на рис. 30. В нашем приложении, таким образом, необходимо для всех отображаемых таблиц (кроме тех, которые используются в подстановочных полях) создать компоненты типа TDataSource и связать их с таблицами с помощью их свойства Dataset.

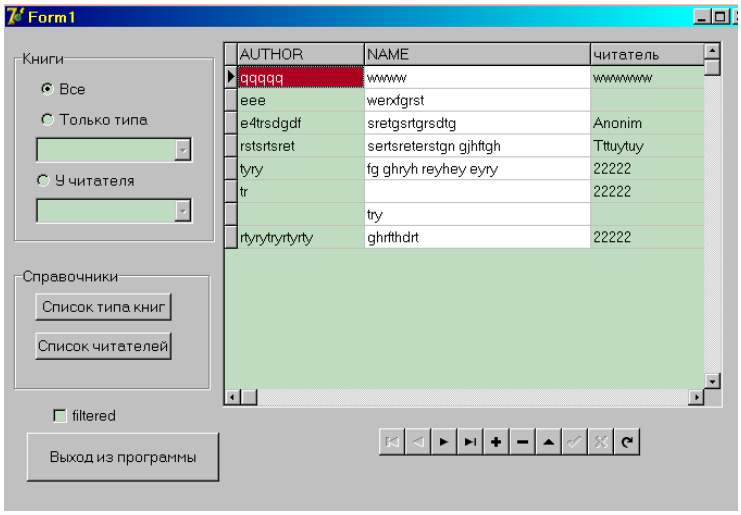


Рис. 6.5. Внешний вид главной формы приложения

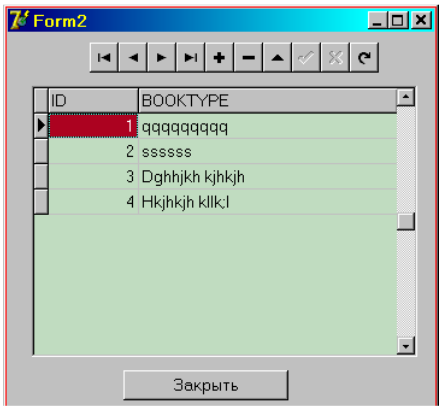


Рис. 6.6. Форма для редактирования справочника типов книг

В главной форме нашего приложения необходимо разместить компоненты с соответствии с рис. 31. В правой части надо разместить компонент TDBGrid для отображения таблицы в виде сетки. С его помощью можно редактировать содержимое таблицы, добавлять новые записи с помощью клавиши Insert, а также удалять записи с помощью комбинации клавиш Ctrl+Delete. В верхней части формы необходимо разместить компонент TDBNavigator, помогающий перемещаться по таблице, редактировать ее, вставлять и удалять записи. Обоим этим компонентам надо указать ис-. точник данных - свойство DataSource - с помощью выпадающего списка в инспекторе объектов. Но т.к. наши источники данных находятся в другом модуле, то перед этим необходимо указать, что данный модуль (главная форма) использует другой (модуль данных). Это делается в Delphi с помощью команды меню File|Use Unit..., которая по сути добавляет в секцию uses раздела implementation выбранный модуль проекта.

В левой части главной формы необходимо разместить кнопку выхода из программы и две кнопки для редактирования справочников, которые просто должны выводить на экран две другие формы простого содержания, как на рис. 6.5. В этих формах также необходимо указать, что надо использовать модуль данных Data, и затем просто выставить свойства DataSource.

Обратите внимание, что в этих формах видно только одно строковое поле, а поле ID скрыто. Просто у этого поля в редакторе списка полей свойство Visible установлено равным False.

В левой верхней части формы необходимо разместить компоненты, позволяющие фильтровать записи в таблице в соответствии с некоторым критерием. Нам необходимо сделать три варианта: все книги, книги заданного типа и книги, находящиеся у заданного пользователя. Для извлечения данных из таблиц в соответствии с некоторым критерием в Delphi возможны несколько вариантов.

Один из вариантов заключается в задании выражения фильтра на записи в свойствах Filter и Filtered компонента TTable. Например, выражение фильтра «Type<>l and Author='Стругац*'» для главной таблицы нашего приложения позволяет выбрать только те книги, код которых отличается от 1 и имя автора начинается с букв ' Стругац'.

Другой вариант заключается в создании связки таблиц «главная-подчиненная» (master-detail) по некоторым полям. Тогда при выборе записи в главной таблице в подчиненной таблице будут отображаться только те, код связи которых совпадает с текущей записью в главной. Эта связка аналогична той, что используется в подстановочных полях: там главной таблицей являлась TableBcoks, а подчиненными - справочники TableTypes HTableUsers.

В данном случае необходимо, чтобы, наоборот, таблицы TableTypes и Tablellsers были главными, а TableBooks - подчиненной. При выборе типа книги или пользователя в таблице книг должны выбираться только определенные записи. Для организации такой связи а подчиненной таблице необходимо указать три свойства, краткое описание которых приведено в табл. 24.

Таблица 24. Свойства компонентов типа TTable для организации связи с главной таблицей в режиме подчиненной

Свойство	Тип	Комментарий .
MasterSource	TDataSource	Источник данных с главной таблицей
IndexFieldNames	String	Имя индексированного поля подчиненной таблицы, по которому осуществляется связь
MasterFields	String	Имя поля главной таблицы, текущее значение которого используется для фильтрации записей в подчиненной

Итак, вернемся к нашему приложению. В левой верхней части главной формы следует разместить три радио-кнопки, при выборе которых необходимо динамически менять свойства MasterSource, IndexFieldNames и MasterFields компонента TableBooks. Для выбора типа книг или пользователя для фильтрации необходимо разместить около радио-кнопок компоненты типа TDBLookupComboBox, при раскрытии которых в ниспадающем списке будут отображаться записи таблиц TableTypes и TableUsers. Для этого нужно у компонентов TDBLookupComboBox установить свойства ListSource в значение DataSourceType (или DataSourceUsers); ListFields равным «Booktype» (или «UserName») и свойство KeyField равным ID.

Лабораторная работа № 7. Создание справочной службы Windows-программы.

<u>Цель работы</u>: получение навыков создания средств поддержки пользователя.

Теоретический материал для выполнения работы.

Создание справочной системы

В состав поставки Borland Delphi 7.0 входят средства создания встроенной контекстно-зависимой справочной службы для создаваемых прикладных программ. Вид окон для отображения справок и правила пользования этими окнами стандартны для Windows. Главное окно Help-службы содержит стандартное' меню с опциями "Файл", "Редактирование" и т.д. и инструментальную строку с кнопками "Содержание", "Поиск", "Хронология" и др. Текст справок может содержать перекрестные ссылки, по которым осуществляется переход к другим темам.

При разработке справочной системы необходимо выполнить следующие шаги:

7.1. Планирование системы справок.

На этом этапе составляется перечень разделов (тем) и необходимых перекрестных ссылок, составляются тексты тем.

7.2. Создание текстового файла

Создание текстового файла (файлов) содержащего описания справочных тем. Текстовый файл готовится с помощью любого текстового редактора, поддерживающего формат RTF - Rich Text Format. Это может быть редактор Word for Windows (WinWord). Формат RTF позволяет вставлять в текст специальные управляющие символы.

Текстовый файл содержит не только тексты отдельных тем, но и дополнительную информацию, а именно:

- названия тем. Каждая тема может иметь название (не путайте с заголовком темы, который присутствует в тексте темы). Эти названия появляются после активизации опции "Закладка" в меню Help-службы. Они также указываются в списке тем кнопок "Поиск" и "Хронология";
- ключевые слова. С их помощью можно искать нужную тему в диалоговом окне, открывающемся кнопкой "Поиск" (в нем имеется строка для ввода ключевых слов, или названий тем). Для любой темы можно назначить сколько угодно ключевых слов, одно слово может быть связано с несколькими темами;
- идентифицирующие строки. Каждая тема, доступная с помощью перекрестных ссылок или индексных указателей, должна иметь связанную с ней уникальную текстовую строку (идентификатор);
- перекрестные ссылки. Они представляют собой выделенную особым цветом и подчеркиванием часть текста, выбор которой приводит к смене темы.

В текстовый файл включаются такие управляющие символы:

Сноска «#»	идентификатор раздела
Перечеркнутый или дважды подчеркнутый текст	Определяет отображение связанного с текстом раздела (перекрестной ссылки) в стандартном справочном окне
Подчеркнутый текст	Определяет отображение перекрестной ссылки в окне пояснений («всплывающем» окне)
Скрытый текст	Определяет идентификатор раздела, связанного с перекрестной ссылкой.
Сноска «$»	Задает название раздела
Сноска «K»	Указывает список ключевых слов
Сноска «+»	Задает порядковый номер раздела в списке просмотра связанных разделов
Сноска «^»	Определяет условие компиляции раздела
Сноска «>»	Определяет тип дополнительного окна, в котором будет отображаться раздел

Сноска «!»	Указывает макрокоманду, которая будет выполняться при открытии окна с разделом.

Задание идентифицирующей строки и организация перекрестных ссылок. Для задания перекрестных ссылок, реализующих скачок от одного раздела к другому, разделы помечаются уникальными *идентифицирующими строками* (идентификаторами). Только помеченные идентификаторами разделы можно просматривать в рамках гипертекстовой системы. Идентифицирующая строка может содержать любые символы, кроме #, @, !, *, =, >, % и пробелов. Длина строки до 255 символов. В качестве идентификаторов имеет смысл использовать текст заголовков раздела, в котором пробелы заменены символами подчеркивания. *Идентификатор* задается с помощью символа «#» в самом начале раздела. Например:

Назначение_программы

Перекрестная ссылка представляет собой выделенную цветом часть текста, щелок мышью по которой приводит к смене раздела. Для кодирования ссылки она выделяется в тексте перечеркнутым или дважды подчеркнутым шрифтом и сразу за ней без каких-либо пробелов указывается идентификатор темы в виде скрытого текста.

Любой вызываемый раздел может отображаться в основном *окне справки* или в *окне пояснений*. Это окно появляется поверх основного окна и обычно используется для пояснений.

В выделенную скрытым текстом часть ссылки можно вставить следующие дополнительные управляющие символы:

* отменяет выделение текста ссылки цветом;

% отменяет выделение текста ссылки цветом подчеркиванием;

@ указывает *HLP*-файл, в которой расположена нужная тема;

> указывает тип окна для отображения раздела.

Символы * и % вставляются непосредственно перед идентификатором раздела и, так же как и он, оформляются скрытым текстом. Например:

~~ГИПЕРТЕКСТ~~

В ссылке можно указать только один из символов отмены выделения «*», «%».

Символы «@» «>», наоборот, вставляются в конце скрытого текста, и за ними должны следовать:

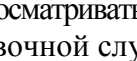
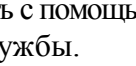
- маршрут доступа и имя *HLP*-файла – для символа «@»;
- имя типа окна, в котором следует отобразить раздел – для символа «>»; это окно должно определяться в файле проекта. Имя окна нужно указывать заглавными буквами, даже если в файле проекта оно определено строчными.

Название раздела используется в *Help*-службе после активизации опции *Закладка* в главном окне; и в списке разделов диалоговых окон, связанных с кнопками *Поиск* и *Хронология* инструментальной панели справочного окна. Название раздела задается с помощью сноски «$», которая должна предшествовать первому символу текста раздела. Название раздела записывается в тексте сноски и отделяется от символа «$» одним пробелом.

Help-служба позволяет искать разделы по связанным с ними *ключевым словам*. Для определения ключевого поля в начале раздела (до первого символа текста раздела) ставится сноска, помеченная латинской буквой «K» или «k». Например:

K открыть; текст файл; ASCII; текст

Вся связанные с разделом ключевые слова помещаются в текст сноски и отделяются от «K» одним пробелом, а друг от друга символом «;». Группы связанных по смыслу ключевых слов объединяются во фразы, которые отделяются друг от друга пробелами. *Help*-служба ищет и отображает в списке выбора названия всех разделов, ключевые слова которых перечислены в одной фразе.

Порядок просмотра связанных разделов. Связь разделов подразумевается создателем и может быть любая – по смысловой связанности, в алфавитном порядке, в порядке знакомства с программным продуктом и т.д. Связанные разделы можно просматривать с помощью кнопок  и  , находящихся на инструментальной панели справочной службы.

Чтобы связать с разделом код для указания порядка просмотра раздела, необходимо вставить сноску «+»:

+ гл_меню:010

Обычно в качестве кодовой последовательности выбирают общее название связанных разделов и цифры порядкового номера. Номера полезно задавать с некоторым шагом (например, 5), чтобы была возможность дополнения в списки разделов и при этом не изменять уде существующие сноски.

7.3. Разработка проектного файла

Проектный файл служит основным управляющим документов для *Help*-компилятором. В *Windows 32* он создается с помощью утилиты *Microsoft Help Workshop (MS HW)* и представляет собой *ASCII*-текстовый файл, содержащий несколько секций. Для запуска *MS HW* следует загрузить файл HELP | TOOLS | HCW.EXE из папки *Delphi*.

Проектный файл должен иметь расширение .HPJ. Он состоит из нескольких секций. Секция – это фрагмент текста, состоящий из заголовка и одной или нескольких следующих за ним строк (опций) вида

ИМЯ_ПАРАМЕТРА = ЗНАЧЕНИЕ.

При этом обязательной является только секция [Files]. В ней указываются имена текстовых файлов со справочной информацией. Например:

```
[Files]
myhelp.rtf
```

В остальных секциях проектного файла указываются: каталог, содержащий текстовый файл: имя шрифта, используемого справочной службой; идентификаторы справочных тем и др.

7.4. Файл содержания справочной службы

Содержание справочной службы оформляется в файле с расширением *CNT* и становится доступным после нажатия кнопки *Содержание* в справочном окне. Для создания файла содержания также используется утилита *MS HW*.

7. 5. Компиляция проектного файла и получение ресурсного файла справочной службы.

Для компиляции проектного файла нужно щелкнуть по кнопке *Save and Compile.* После окончания компиляции, отображается окно компилятора с указанием предупреждений, замечаний и критических ошибок.

Результатом прогона компилятора является ресурсный файл с расширением hlp (например - myhelp.hlp).

Тестирование файла содержания заключается в автоматическом вызове всех указанных в содержании разделов. Если какой-либо раздел не вызывается, выдается сообщение, позволяющее найти устранить ошибку.

7.6. Связь с программой.

Связь с программой реализуется с помощью свойств *HelpContext* видимых компонентов, в которые следует поместить числовые идентификаторы нужных разделов справочной службы так, как они определены в секции *MAP*. Кроме того, в свойство Application.HelpFile нужно поместить имя *HLP*-файла. После такой настройки пользователь программы сможет с помощью клавиши *F*1 получить контекстно-чувствительную справку.

Чтобы прикладная программа вызывала справочную службу из меню, в ней необходимо предусмотреть вызов специальной API-функции WinHelp:

```
Function WinHelp (Wnd:hWnd; HelpFile:Pchar; Command:Word;   Data: LongInt): Bool;
```

Wnd - дескриптор окна, из которого вызывается справка (свойство Handle);
HelpFile - имя HLP-файла;
Command - команда, чаще всего используются следующие:

- help_context - отображает тему, указанную параметром Data;
- help_contents - показывает тему с оглавлением;

help_key - по ключевому слову, указанному в Data;
Data - зависит от команды, обычно указывает тему.

Лабораторная работа № 8. Создание прототипов интерфейсов. Количественный анализ интерфейсов.

<u>Цель работы</u>: получение навыков квантификации интерфейса на основе модели GOMS.

Теоретический материал для выполнения работы.

Количественный анализ интерфейса

Одним из лучших подходов к количественному анализу моделей интерфейсов является классическая модель GOMS — «правила для целей, объектов, методов и выделения» (the model of goals, objects, methods and selection rules). Моделирование GOMS позволяет предсказать, сколько времени потребуется опытному пользователю на выполнение конкретной операции при использовании данной модели интерфейса.

Временные интервалы в интерфейсе

Время, требующееся для выполнения какой-то задачи системой «пользователь-компьютер», является суммой всех временных интервалов, которые потребовались системе на выполнение последовательности элементарных жестов, составляющих данную задачу. С помощью тщательных лабораторных исследований был получен набор временных интервалов, требуемых для выполнения различных жестов.

Номенклатура временных интервалов при квантификации интерфейса

K = 0.2 c	Нажатие клавиши. Время, необходимое для того, чтобы нажать клавишу.
P = 1.1 c	Перемещение курсора мыши. Время, необходимое пользователю для того, чтобы указать на какую-то позицию на экране монитора.
H = 0.4 c	Перемещение. Время, необходимое пользователю для того, чтобы переместить руку с клавиатуры на мышь или с мыши на клавиатуру.
M = 1.35 c	Продолжительность выбора действия. В среднем, за 1.2 секунды пользователь принимает решение, какое именно действие он должен совершить на следующем шаге. Обычно это самый сложный оператор, поскольку часто непонятно, в каких именно местах процедуры его необходимо ставить. Например, иногда, когда пользователь совершал искомую последовательность действий не раз и при этом совершенно уверен в том, что общий ход процедуры не будет отличаться от обычного, это время затрачивается только в самом начале выполнения (далее действия будут совершаться автоматически). С другой стороны, начинающим пользователям приходится выбирать действие перед каждым своим шагом. Однако в большинстве случаев достаточно считать, что это время нужно добавлять перед всеми нажатиями, которые не приходятся на область с установленным фокусом, перед всеми командами, инициированными мышью и после существенных изменений изображения на экране (но и здравый смысл тут не помешает). С практической точки зрения важнее устанавливать этот оператор везде одинаково, нежели устанавливать его возможно более точно.
R	Ответ. Время, в течение которого пользователь должен ожидать ответ компьютера. От 0,1 сек до бесконечности.

На практике указанные значения могут варьироваться в широких пределах. Для опытного пользователя, способного печатать со скоростью 135 слов/мин., значение K может составлять 0.08 с, для обычного пользователя, имеющего скорость 55 слов/мин., — 0.2 с, для среднего неопытного пользователя, имеющего скорость 40 слов/мин., — 0.28с, а для начинающего — 1.2 с.

Расчеты по модели GOMS

Вычисления времени, необходимого на выполнение того или иного действия (например, «переместить руку с мыши на клавиатуру и набрать букву», с помощью модели GOMS начинаются с перечисления операций из списка жестов модели GOMS, которые составляют это действие. Перечисление движений (K, P и H) — это довольно простая часть модели GOMS. Более сложным, например, в модели скорости печати GOMS, является определение точек, в которых пользователь остановится, чтобы выполнить бессознательную ментальную операцию, — интервалы ментальной подготовки, которые обозначаются символом *M*. Основные правила, позволяющие определить, в какие моменты будут проходить ментальные операции, представлены в таблице.

Правила расстановки ментальных операций

Правило 0 Начальная расстановка операторов *M*	Операторы *M* следует устанавливать перед всеми операторами *K* (нажатие клавиши), а также перед всеми операторами *P* (указание с помощью мыши), предназначенными для выбора команд; но перед операторами *P*, предназначенными для указания на аргументы этих команд, ставить оператор *M* не следует
Правило 1 Удаление ожидаемых операторов *M*	Если оператор, следующий за оператором *M*, является полностью ожидаемым с точки зрения оператора, предшествующего *M*, то этот оператор *M* может быть удален. Например, если вы перемещаете мышь с намерением нажать ее кнопку по достижении цели движения, то в соответствии с этим правилом следует удалить оператор *M,* устанавливаемый по правилу **0**. В этом случае последовательность *P M K* превращается в *P K*.
Правило 2 Удаление операторов *M* внутри когнитивных единиц	Если строка вида *M K M K M K*...принадлежит когнитивной единице, то следует удалить все операторы *M*, кроме первого. Когнитивной единицей является непрерывная последовательность вводимых символов, которые могут образовывать название команды или аргумент. Например, Y, *перемещать, Елена Троянская* или *4564.23* являются примерами когнитивных единиц.
Правило 3 Удаление операторов *M* перед последовательными разделителями	Если оператор K означает лишний разделитель, стоящий в конце когнитивной единицы (например, разделитель команды, следующий сразу за разделителем аргумента этой команды), то следует удалять оператор *M,* стоящий перед ним.
Правило 4 Удаление операторов *M,* которые являются прерывателями команд	Если оператор *K* является разделителем, стоящим после постоянной строки (например, название команды или любая последовательность символов, которая каждый раз вводится в неизменном виде), то следует удалить оператор *M,* стоящий перед ним. (Добавление разделителя станет привычным действием, и поэтому разделитель станет частью строки и не будет требовать специального оператора *M*). Но если оператор *K* является разделителем для строки аргументов или любой другой изменяемой строки, то оператор *M* следует сохранить перед ним.
Правило 5 Удаление перекрывающих операторов *M*	Любую часть оператора *M,* которая перекрывает оператор *R*, означающий задержку, связанную с ожиданием ответа компьютера, учитывать не следует.

Отметим, что в этих правилах под **строкой** будем понимать некоторую последовательность символов. **Разделителем** будет считаться символ, которым обозначено начало или конец значимого фрагмента текста, такого как, например, слово естественного языка или телефонный номер. Например, пробелы являются разделителями для большинства слов. Точка является

наиболее распространенным разделителем, который используется в конце предложений. Скобки используются для ограничения пояснений и замечаний и т.д. Операторами являются *K, P* и *H*. Если для выполнения команды требуется дополнительная информация (как, например, в случае когда для установки будильника пользователю требуется указать время его включения), эта информация называется **аргументом** данной команды.

Примеры расчетов по модели GOMS

Разработка интерфейса обычно начинается с определения задачи или набора задач, для которых продукт предназначен. Суть задачи, а также средства, имеющиеся для реализации ее решения, часто формулируют в виде требования или спецификации.

Задача рассматриваемых прототипов: перевести температурные показания из шкалы Фаренгейта в шкалу Цельсия или наоборот.

Интерфейс 1. Диалоговое окно

Инструкции в диалоговом окне (рис.1) довольно просты. На их основе можно описать метод действий, который должен использовать пользователь в терминах жестов модели GOMS. Запись по модели GOMS будет представлена последовательно по мере того, как будут добавляться новые жесты.

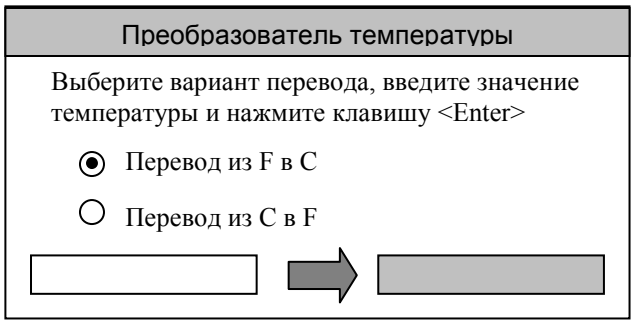


Рис. 1 Вариант диалогового окна с использованием группы переключателей

- Перемещение руки к графическому устройству ввода данных: *H*
- Перемещение курсора к необходимому переключателю в группе: *H P*
- Нажатие на необходимый переключатель: *H P K*

В половине случаев в интерфейсе уже будет выбрано требуемое направление перевода, и поэтому не придется кликать на переключатель. Сейчас мы рассматриваем случай, когда переключатель не установлен в требуемое положение.

- Перемещение рук снова к клавиатуре: *H P K H*
- Ввод четырех символов: *H P K H K K K K*
- Нажатие клавиши <Enter>: *H P K H K K K K K*

Нажатие клавиши <Enter> завершает часть анализа, касающуюся метода. В соответствии с правилом **0** мы ставим оператор *M* перед всеми операторами *K* и *P* за исключением операторов P, указывающих на аргументы, которых в нижеследующем примере нет: *H M P M K H M K M K M K M K M K*

Правило **1** предписывает заменить *P M* K на *P K* и удалить все другие операторы *M,* которые являются ожидаемыми (в указанном примере таких нет). Кроме того, правило **2** предписывает удалять операторы *M* в середине цепочек. После применения этих двух правил остается следующая запись: *H M P K H M K K K K M K.*

В соответствии с правилом **4** следует оставить оператор *M* перед конечным *K.* Правила **3** и **5** в данном примере не применяются. Следующий шаг — это заменить символы операторов на соответствующие временные интервалы (напомним, что *K* = 0.2; *P* = 1.1; *H* = 0.4; *M* = 1.35).

H + M + P + K + H + M + K + K + K + K + M + K = 0.4 + 1.35 + 1.1 + 0.2 + 0.4 + 1.35 + 4 * (0.2) + 1.35 + 0.2 = 7.15 с

В случае, когда переключатель уже установлен в требуемое положение, метод действий становится следующим:

M K K K K M K

M + K + K + K + K + M + K = 3.7c

По условиям задачи оба случая являются равновероятными. Таким образом, среднее время, которое потребуется на использование интерфейса для перевода из одной шкалы в другую, составит (7.15 + 3.7)/ 2 ~ 5.4 с.

Далее мы рассмотрим графический интерфейс, в котором используется известная всем метафора.

Интерфейс 2. ГИП (GUI, graphical user interface)

В интерфейсе, показанном на рис. 2, используется наглядное отображение термометров. Пользователь может поднять или опустить указатель на каждом термометре методом перетаскивания с помощью мыши. В этом интерфейсе не требуется вводить символы посредством клавиатуры — пользователь просто выбирает значение температуры на одном из термометров. При перемещении указателя на одном термометре указатель на другом перемещается на соответствующее значение. Точность устанавливается с помощью регуляторов масштабирования шкал. Также возможно изменить текущий диапазон значений. Изменение шкалы или диапазона на одном термометре автоматически приводит к соответствующему изменению на другом. Точное числовое значение отображается на перемещаемой стрелке.

С помощью нажатия кнопок «Расширить шкалу» и «Сжать шкалу» можно уменьшить или увеличить цену деления шкал в 10 раз.

Провести анализ этого графического интерфейса с помощью модели скорости печати GOMS довольно сложно, поскольку способ, которым пользователь может его использовать, зависит от того, где в данный момент установлена стрелка указателя, какой необходим диапазон температур и какая требуется точность. Рассмотрим сначала простой случай, при котором диапазон температурных шкал и точность перевода уже находятся в желаемом положении. Анализ позволит определить минимальное время, необходимое для использования этого интерфейса.

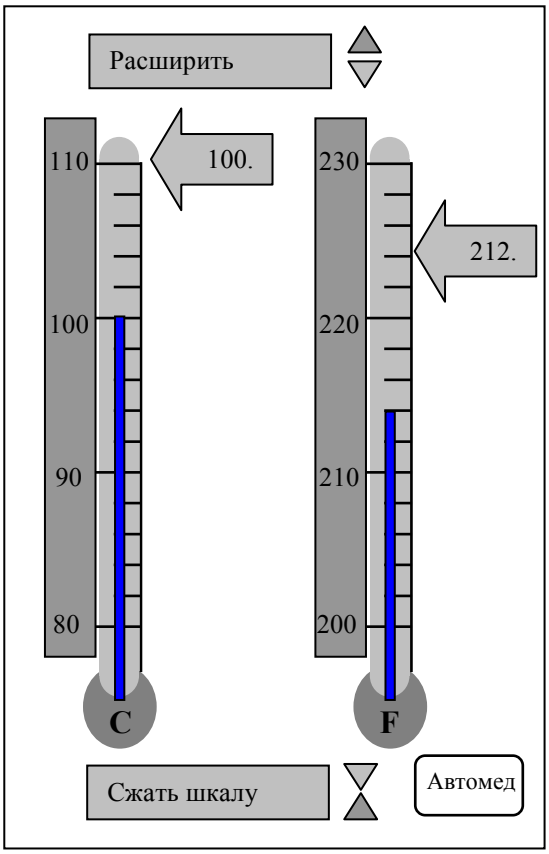


Рис. 2 Графический интерфейс

• Запишем, какие жесты использует пользователь, когда перемещает руку к ГУВ, щелкает по кнопке и удерживает ее, указывая на стрелку одного из термометров: *Н Р К*

• Продолжим записывать те жесты, которые используются для перемещения стрелки к необходимому температурному значению и отпускает кнопку ГУВ: *Н Р К Р К*

• Поставим операторы *М* в соответствии с правилом **0**: *Н М Р М К М К*

• Удалим два оператора *М* в соответствии с правилом **1**: *Н М Р К К*

Когнитивные единицы, разделители последовательностей и т. д. здесь не используются, поэтому правила **2-5** не применяем. Складывая значения операторов, получаем общее время:
Н М Р К К

0.4 + 1.35 + 1.1 + 0.2 + 0.2 = 3.25 с

Результат вычисления относится к удачному случаю, когда исходный термометр уже предустановлен на требуемый диапазон и точность. Теперь рассмотрим случай, при котором приходится расширять шкалу, чтобы увидеть необходимое температурное значение, изменять диапазон, сжимать шкалу, чтобы получить требуемую точность, и затем перемещать стрелку указателя. Далее приводится общая запись метода без промежуточных шагов.
Н Р К S К Р К S К Р К S К Р К К
В соответствии с правилами расставляем операторы М:
Н + 3(М + Р + К + S + К) + М + Р + К + К
0.4 + 3 * (1.35 + 0.2 + 3.0 + 0.2) + 1.35 + 0.4 + 0.2 + 0.2 - 16.8 с

За исключением редких случаев, когда шкалы уже с самого начала установлены правильно, идеальному пользователю понадобится более 16 с на то, чтобы выполнить перевод из одной шкалы в другую, тогда как реальный, т. е. не идеальный пользователь, может сбивать шкалы и стрелки указателей, и поэтому ему понадобится даже больше времени.

Задание.

1. Создать два прототипа интерфейса для решения предложенной задачи.
2. Выполнить количественную оценку интерфейсов.

Список литературы

1. Акчурин Э.А. Человеко-машинное взаимодействие: Учебное пособие. —М.: СОЛОН-ПРЕСС, 2008.-93 с. (
2. Логунова О.С., Ячиков И.М., Ильина Е.А. Человеко-машинное взаимодействие: Теория и практика. — Ростов-на-Дону: «Феникс», 2006.-288 с.