Федеральное государственное бюджетное образовательное учреждение высшего образования
«Бурятская государственная сельскохозяйственная академия имени В.Р. Филиппова»

Экономический факультет

«СОГЛАСОВАНО»	«УТВЕРЖДЕНО»
Заведующий выпускающей кафедрой Информатика и информационные технологии в экономике	Декан Экономический факультет
к.ф.-м.н., доцент	к.э.н., доцент
Садуев Н.Б.	Баниева М.А.
подпись	подпись
«23» января 2025	«23» января 2025

Оценочные материалы
дисциплины (модуля)
Б1.В.13 Проектирование мобильных приложений
Направление подготовки
09.03.03 Прикладная информатика
Направленность (профиль)
Прикладная информатика в экономике АПК

Улан-Удэ, 2025г.

ВВЕДЕНИЕ

1. Оценочные материалы по дисциплине (модулю) являются обязательным обособленным приложением к Рабочей программе дисциплины (модуля) и представлены в виде оценочных средств.
2. Оценочные материалы является составной частью нормативно-методического обеспечения системы оценки качества освоения обучающимися указанной дисциплины (модуля).
3. При помощи оценочных материалов осуществляется контроль и управление процессом формирования обучающимися компетенций, из числа предусмотренных ФГОС ВО в качестве результатов освоения дисциплины (модуля).
4. Оценочные материалы по дисциплине (модулю) включают в себя:
- оценочные средства, применяемые при промежуточной аттестации по итогам изучения дисциплины (модуля).
- оценочные средства, применяемые в рамках индивидуализации выполнения, контроля фиксированных видов ВАРО;
- оценочные средства, применяемые для текущего контроля;
5. Разработчиками оценочных материалов по дисциплине (модулю) являются преподаватели кафедры, обеспечивающей изучение обучающимися дисциплины (модуля), в Академии. Содержательной основой для разработки оценочных материалов является Рабочая программа дисциплины (модуля).

Перечень видов оценочных средств

1. Перечень вопросов к экзамену
2. Комплект заданий для лабораторных работ
3. Перечень контрольных вопросов для проведения устных опросов
4. Перечень вопросов для самостоятельного изучения
5. Тестовые задания
6. Кейс-задания

Средства для промежуточной аттестации по итогам изучения дисциплины

Нормативная база проведения промежуточной аттестации обучающихся по результатам изучения дисциплины: Проектирование мобильных приложений

1) действующее «Положение о текущем контроле успеваемости и промежуточной аттестации обучающихся ФГБОУ ВО Бурятская ГСХА»

Основные характеристикипромежуточной аттестации обучающихся по итогам изучения дисциплины (модуля)	
1	2
Цель промежуточной аттестации -	установление уровня достижения каждым обучающимся целей обучения по данной дисциплине
Форма промежуточной аттестации -	Экзамен
Место экзамена в графике учебного процесса:	1) подготовка к экзамену и сдача экзамена осуществляется за счёт учебного времени (трудоёмкости), отведённого на экзаменационную сессию для обучающихся, сроки которой устанавливаются приказом по академии
	2) дата, время и место проведения экзамена определяется графиком сдачи экзаменов, утверждаемым деканом факультета (директором института)
Форма экзамена -	(Письменный, устный)
Процедура проведения экзамена -	представлена в оценочных материалах по дисциплине
Экзаменационная программа по учебной дисциплине:	1) представлена в оценочных материалах по дисциплине 2) охватывает все разделы дисциплины

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

Контрольные вопросы и задания для проведения текущего контроля и промежуточной аттестации по итогам

Перечень вопросов к экзамену

1. Какое событие генерируется на этапе выгрузки страницы? ПК-2, ПК-5
2. Какие языки разметки поддерживаются управляющими элементами ASP.NET Mobile Controls? ПК-2, ПК-5
3. Из каких компонентов состоит значение атрибута? ПК-2, ПК-5
4. Какой режим работы устройства Bluetooth предусматривает регулярные обмены с целью синхронизации клиентов? ПК-2, ПК-5
5. Что является недостатком беспроводных сетей? ПК-2, ПК-5
6. Что такое том в контексте использования EDB или CEDB? ПК-2, ПК-5
7. Какие данные являются пользовательскими данными? ПК-2, ПК-5
8. Какой компонент сети GPRS является блоком управления пакетами, дающим возможность станциям GSM пересылать и получать пакеты при GPRS коммуникациях?
9. Что описывает модель доступа к данным? ПК-2, ПК-5
10. Что является достоинством локальных баз данных? ПК-2, ПК-5
11. Какие элементы управления являются списочными? ПК-2, ПК-5
12. В каких случаях можно создать журнал ошибок? ПК-2, ПК-5
13. Чем определяется выбор наиболее подходящей модели хранения данных в памяти? ПК-2, ПК-5
14. В каком случае исполнительная среда должна найти сборку, которая содержит требуемый тип, и загрузить его? ПК-2, ПК-5

15. Какой вариант установки приложения на мобильное устройство является аналогом установки программного обеспечения на настольных компьютерах с компакт-дисков, когда инсталляционная программа автоматически запускается после вставки компакт-диска или диска DVD в соответствующий привод? ПК-2, ПК-5

16. Сколько активных узлов может поддерживать главный узел пикосети? ПК-2, ПК-5

17. Какие недостатки имеют устройства с сенсорным экраном? ПК-2, ПК-5

Комплект заданий для лабораторных работ

Лабораторная работа №1.Мобильные устройства и их характеристики. Платформа Windows Mobile.

Целью данной лабораторной работы является знакомство с мобильными элементами управления и их возможностями.

Задание на лабораторную работу №1.

- Выбрать предметную область, проблему, для решения которой необходимо создание мобильного IT - решения. Разработка примерной архитектуры приложения.
- Рассмотреть имеющиеся элементы управления, возможности их применения, методы, события.
- Создать мобильную форму с простейшим кодом (лучше всего для данных целей выбрать задачу, связанную с установлением соединения с БД, редактирования записей, вывод результатов простейших запросов, авторизацию и т.п.)

1. Эмулятор мобильных устройств.

Как и Visual Studio 2005, версия Visual Studio 2008 Professional пригодна для проектирования приложений для мобильных устройств с использованием компонента SmartDevice Projects (функции разработки для мобильных устройств в Visual Studio 2008 Standard Edition отсутствуют). В новом эмуляторе устройств Visual Studio 2008 появилось несколько улучшений. Можно проектировать программы для Pocket PC 2003, Smartphone 2003 и WindowsMobile 5.0. Благодаря эмуляции батарей, эмулятор устройств может выдать событие низкого заряда батарей. Таким образом, мобильные приложения могут проверять заряд батарей.

Рис. 1. Эмулятор мобильных устройств Visual Studio 2008.

В состав Visual Studio 2008 входит новейшая версия платформ разработки устройств Microsoft:.NET Compact Framework 3.5. Можно задействовать различные версии.NET Compact Framework. Таким образом, при создании нового проекта SmartDevice можно выбрать в качестве целевой платформы .NET Compact Framework 2.0 или .NET Compact Framework 3.5.

Кроме того, Visual Studio 2008 предоставляет новые службы Sync Services for ADO. NET, через которые новые мобильные приложения обеспечивают конечным пользователям одинаково удобную работу как с подключенным, так и с отключенным от сети устройством. Мобильное устройство с локальным хранилищем SQL Server Compact и службами Sync Services может периодически подключаться к серверу базы данных и выполнять двунаправленную синхронизацию с мобильным приложением.

2. Краткий обзор мобильных элементов управления.

2.1.Списочные элементы управления.

Списочный элемент управления направляет на устройство последовательность элементов. Он может работать в статическом или интерактивном режиме. В статическом режиме данный элемент генерирует статический список чисто текстовых элементов. В интерактивном режиме получаются элементы, по щелчку на которых генерируются события.

2.2.Элемент управления ListBox.

Представляет собой список с полосой прокрутки.

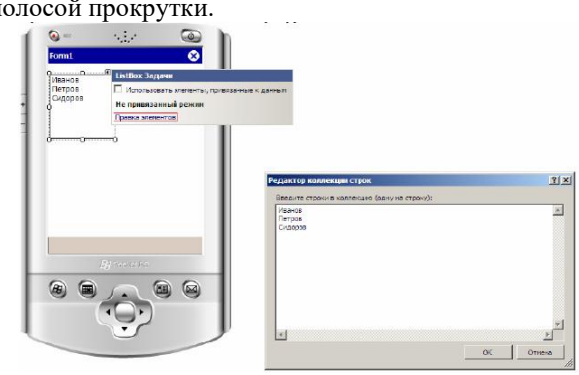


Рис. 2. Редактирование элементов ListBox.

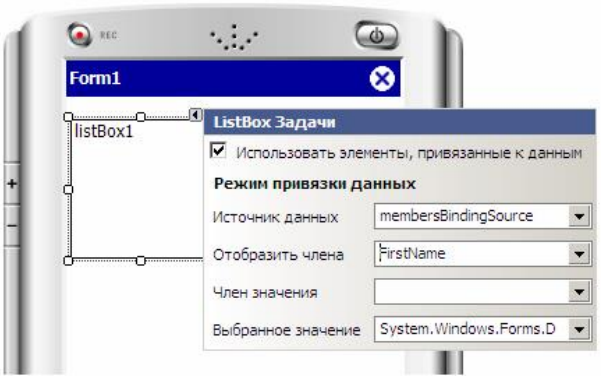


Рис. 3. Привязка ListBox'а к источнику данных.

2.3.Элемент управления ComboBox.

Представляет собой выпадающий список

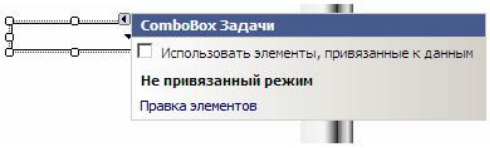


Рис. 4. Задачи по привязке или заданию данных в элементе управления ComboBox.

2.4.Элемент управления TextBox.

Элемент управления TextBox генерирует однострочное текстовое поле и хранит в свойстве Text введенный пользователем текст. Данный элемент может работать в режиме пароля и в числовом режиме, но не поддерживает многострочного редактирования и не может выводиться как поле, доступное только для чтения. Вы можете задавать для него способ выравнивания и максимальную длину.

2.5.Элемент управления InputPanel.

Обеспечивает управляемую реализацию элемента управления "программная панель ввода" в Windows Embedded CE. При создании приложения Windows, предназначенного для платформы Pocket PC, среда разработки Visual Studio автоматически добавляет компонент MainMenu в файл проекта для Form1. Этот компонент добавляет на форму нижнюю строку меню со значком программной панели ввода. Для использования программной панели ввода не требуется программирования: пользователи могут включать и отключать её отображение с помощью значка. Для программирования программной панели ввода необходимо перетащить компонент InputPanel с панели элементов ToolBox на форму.

Программную панель ввода можно показать программно с помощью свойства Enabled, получить её размер - с помощью свойства Bounds, и определить размер области формы, не занятый этой панелью, - с помощью свойства VisibleDesktop. Обратите внимание, что для устройств Pocket PC свойство Bounds всегда возвращает ширину, равную 240 точкам, и высоту, равную 80 точкам, независимо от того, включена ли программная панель ввода. Событие EnabledChanged происходит каждый раз при включении или выключении программной панели ввода, как пользователем, так и программно. Обычно причиной для программирования программной панели ввода является изменение положения или размера элементов управления на форме, когда эта панель включена и отключена.

В следующем примере на странице вкладки содержится текстовое поле. Программная панель ввода отображается на вкладке, когда текстовое поле получает фокус, при этом используется событие EnabledChanged, чтобы уменьшить высоту вкладки, когда эта панель включена, и восстановить исходную высоту, когда она выключена.

```
private void textBox1_GotFocus(object sender, System.EventArgs e)
{
// Отображение InputPanel
inputPanel1.Enabled = true;
}
private void inputPanel1_EnabledChanged(object sender, EventArgs e)
{
if (inputPanel1.Enabled == false)
{
// Поскольку InputPanel неактивна, высота вкладки устанавливается в
// исходное положение
VisibleRect = inputPanel1.VisibleDesktop;
tabControl1.Height = TabOriginalHeight;
}
else
{
// InputPanel активна, поэтому высота вкладки уменьшается
VisibleRect = inputPanel1.VisibleDesktop;
tabControl1.Height = VisibleRect.Height;
}
// Свойство Bounds всегда возвращает значения высоты 80 и ширины 240 пикселей
// для устройств PocketPC
BoundsRect = inputPanel1.Bounds;
```

```
VisibleInfo.Text = String.Format("VisibleDesktop: X = {0}, " +
"Y = {1}, Width = {2}, Height = {3}", VisibleRect.X,
VisibleRect.Y, VisibleRect.Width, VisibleRect.Height);
BoundsInfo.Text = String.Format("Bounds: X = {0}, Y = {1}, " +
"Width = {2}, Height = {3}", BoundsRect.X, BoundsRect.Y,
BoundsRect.Width, BoundsRect.Height);
}
```

Наиболее полную информацию о мобильных элементах управления можно получить на http://msdn.microsoft.com/ru-ru/library/aa139616.aspx

Лабораторная работа №2. Обзор инструментальных средств разработки приложений для мобильных устройств под управлением платформ windows Mobile.

Цель лабораторной работы - знакомство с основными методами и способами управления памятью при создании мобильного приложения.

Жизненный цикл мобильной страницы.

При проектировании веб - приложений для мобильных устройств важно учитывать особенности жизненного цикла мобильной страницы.

Жизненный цикл мобильной страницы ASP.NET, этапы которого перечислены ниже, практически идентичен жизненному циклу обычной страницы Web Forms. Он включает те же события, хотя поведение системы до и после их генерирования разное.

- *Инициализация страницы*. На этом этапе определяется адаптер устройства, который будет использоваться при формировании вывода страницы, и устанавливается свойство Adapter класса MobilePage. Поиск адаптера начинается с файла machine.config, затем производится в файле web.config из корневого каталога сайта, а уж потом - в файлах web.config, расположенных ниже по иерархии. Адаптер выбирается на основе характеристик текущего устройства. Обратите внимание: с целью повышения производительности адаптер кэшируется, поэтому его поиск для каждого пользовательского агента производится один раз.
- *Загрузка состояния представления*. На данном этапе восстанавливается сохраненное состояние страницы. Никакие события с этим этапом не ассоциированы.
- *Загрузка данных возврата страницы*. Страница загружает входящие данные формы, кэшированные в объекте Request, и соответствующим образом обновляет свои свойства. Никакие пользовательские события с данным этапом не ассоциированы.
- *Загрузка пользовательского кода*. Страница готова к выполнению инициализационного кода, связанного с ее логикой и поведением. Она генерирует событие Load и загружает информацию с учетом специфики адаптера устройства. Если вы хотите управлять этим этапом, то можете реализовать обработчик данного события как для страницы, так и для адаптера.
- *Отправка уведомлений об изменении возвращенных данных формы*. Элементы управления, входящие в состав страницы, генерируют события изменения, если их состояние изменилось со времени последнего возврата формы. Для того чтобы элемент управления мог генерировать на этом этапе событие изменения состояния, в нем нужно реализовать интерфейс IPostBackDataHandler. Пользовательские события на данном этапе не генерируются.
- *Обработка события возврата формы*. Страница выполняет код, связанный с событием, которое вызвало возврат формы.
- *Предрендеринг*. Перед рендерингом вывода код получает последнюю возможность внести какие-либо изменения. Событие, с которым разработчик может связать соответствующий код мобильной страницы или адаптера устройства, называется PreRender. Именно на этом этапе производится разбивка на страницы. Вывод страницы формируется с учетом параметров разбивки.
- *Сохранение состояния представления*. Состояние страницы сериализуется в строку, которая затем сохраняется - обычно в виде скрытого поля. Пользовательские события на данном этапе не генерируются.
- *Рендеринг страницы*. Страница генерирует вывод, который будет направлен клиенту. За вывод дочерних элементов управления в нужном порядке отвечает адаптер.
- *Выгрузка страницы*. На этом этапе адаптер устройства выполняет необходимые финальные операции. Событие Unload доступно и странице, и адаптеру устройства.

Разработка архитектуры приложения.

Многие настольные приложения выполняют поиск инкрементально - пользователь вводит частичный ключ и система возвращает список соответствий. Затем пользователь либо сужает область поиска, либо просто выбирает один из предложенных вариантов. Такой способ работы предполагает наличие удобной клавиатуры, но у мобильных устройств ее обычно не бывает. Как правило, такое устройство снабжено цифровой клавиатурой, но алфавитной - далеко не всегда. А рассчитывать на то, что пользователю понравится постоянно вводить текст, пользуясь цифровой клавиатурой, явно не приходится, особенно если учесть, что мобильные устройства потому и называются мобильными, что ими часто пользуются на ходу или в дороге, работая одной рукой.

Поэтому разработчик мобильного приложения, в особенности предназначенного для сотовых телефонов, должен хорошо усвоить следующее правило: один дополнительный щелчок предпочтительнее набора текста. Для персонализации и оптимизации вывода приложения используйте имеющуюся информацию о пользователе (его имя, адрес электронной почты, профиль или даже местоположение).

Структура формы должна быть как можно более простой и сжатой, с краткими, но понятными описаниями и единственным способом выполнения каждой задачи. Данные следует загружать малыми порциями, поскольку полоса пропускания приложения мала, а работа с приложением должна осуществляться путем прокрутки и нажатия программируемых клавиш, а также числовых клавиш на клавиатуре.

Для мобильных приложений, требующих пользовательского ввода, хорошо подходит интерфейс мастера. Очень удобно древовидное представление заданий и данных, поскольку оно позволяет предложить список вариантов, чтобы пользователь мог выбрать один из них или выйти.

Пример простейшей системы авторизации.

В данной задаче, для простоты реализации, имена учетных записей пользователей и пароли будут храниться в базе данных SQL на локальной машине.

Добавьте на форму следующие элементы управления:

TextBox tbLogin
TextBox tbPassword
Label lLogin
Label lPassword
Button btnAuth
Label lAuthStatus

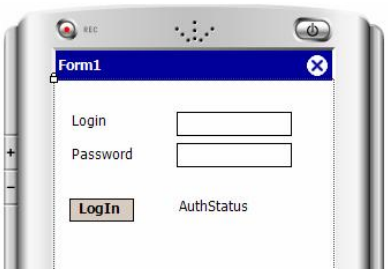


Рис. 5. Образец заполнения формы.

Задайте значение свойства PasswordChar у элемента управления tbPassword (любой символ, который будет служить маской ввода для пароля) (рис.6).

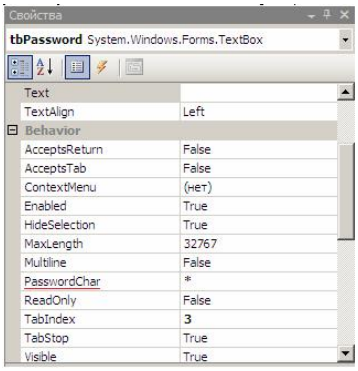


Рис. 6.

На событие Click кнопки btnAuth "определим следующий код"

```
private void btnAuth_Click(object sender, EventArgs e)
{
//задаем свойства подключения к БД
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=N121-6\\SQLEXPRESS;Initial Catalog=mobile;Integrated Security=True";
bool check = false;
try
{
conn.Open();
//определяем запрос для проверки правд доступа учетной записи
SqlCommand cmd = new SqlCommand("Select * From Users");
SqlDataReader dr = cmd.ExecuteReader();
while (dr.Read())
{
if (dr["Login"].ToString() == tbLogin.Text)
{
if (dr["Password"].ToString() == tbPassword.Text)
{
//если проверка имени учетной записи и пароля прошла успешно - выходим из цикла
check = true;
break;
}
}
}
conn.Close();
}
catch (Exception exc)
{
MessageBox.Show("" + exc.Message);
```

```
}
//в случае успешной авторизации выводим соотествующее сообщение
if (check)
{
lAuthStatus.Text = "Авторизация прошла успешно";
lAuthStatus.Visible = true;
```

Лабораторная работа №3. Net Compact Framework и Visual Studio 2008.

Для выполнения данной лабораторной работы необходимо наличие ранее созданного проекта. Изучив теоретические основы управления памятью при создании приложений для мобильных устройств, необходимо применить описанные приемы и методы в созданном приложении (разумеется, при наличии таковой возможности). Результатом лабораторной работы должна являться таблица сравнения результатов выполнения алгоритмов до оптимизации и после, при аналогичной программно - аппаратной базе.

Задание на лабораторную работу

- Ознакомиться с уровнями управления памятью.
- Применить метод Dispose() в целесообразных ситуациях, если в использовании данного метода нет необходимости - обосновать причины такого решения.
- Рассмотреть модели загрузки данных по требованию.
- Разбить программный код приложения на "структуры".
- Оптимизировать работу со строками.
- Составить сравнительную таблицу времени выполнения алгоритмов до и после оптимизации.

1. Уровни управления памятью:

1. Управление памятью на макроскопическом "уровне приложения". Этот уровень относится к данным и ресурсам уровня приложения, которые поддерживаются вашим приложением в процессе выполнения. Эти данные обычно существуют в течение длительного времени, и их область видимости не ограничивается пределами отдельных функций. Для создания эффективно функционирующего мобильного приложения очень важно иметь надежную модель, управляющую объемом данных, подлежащих хранению в памяти в каждый момент времени, и удалением из памяти данных и ресурсов, непосредственное использование которых в ближайшее время не ожидается. Чрезмерный объем долгоживущих данных состояния загромождает память, которую можно было бы использовать для кэширования JIT-компилированного кода или как рабочую память для функций, и заставляет многократно и не самым эффективным образом очищать память от "мусора".

2. Распределение памяти па микроскопическом "уровне алгоритма". Временную память для выполнения команд, определяемых вашими алгоритмами, распределяют функции. Эффективность этого процесса зависит от вашей стратегии реализации алгоритмов. Например, при написании кода, который должен выполняться в циклах, необходимо как можно тщательнее продумывать его эффективность в отношении использования ресурсов, чтобы свести к минимуму непроизводительные накладные расходы. Уделяя пристальное внимание эффективности распределения памяти в создаваемых вами алгоритмах, вы сможете значительно повысить общую производительность приложения.

ОЗУ мобильных устройств имеют гораздо меньший объем и, как правило, не позволяют надлежащим образом реализовать механизмы, использующие дополнительные внешние накопители для организации быстрого обмена страницами с памятью. Кроме того, в условиях ограниченных ресурсных возможностей на мобильных устройствах могут быть реализованы лишь сравнительно простые механизмы очистки памяти от неиспользуемых объектов. Это означает, что небрежно организованное управление памятью в приложениях для мобильных устройств будет иметь весьма заметные отрицательные последствия как на макроскопическом, так и на микроскопическом уровнях. По сравнению с настольными компьютерами мобильные устройства гораздо менее терпимы к любым просчетам в управлении памятью.

Важно также понимать, что, несмотря на то, что каждое следующее поколение устройств по аппаратным характеристикам превосходит предшествующее, нельзя просто "отложить до лучших времен" проблемы управления памятью. Требования к устройствам также возрастают от поколения к поколению, следовательно, актуальность существующих в данный момент проблем в значительной степени сохранится.

Макромодель управления памятью.

Полезно рассортировать данные и ресурсы приложения, с которыми вы работаете, на две разновидности:

- объекты и ресурсы, которые необходимы приложению для эффективного выполнения
- фактическая пользовательская информация, с которой работает приложение.

1. *Необходимые служебные данные приложения*. В эту разновидность входят ресурсы, необходимые приложению для поддержания пользовательского интерфейса и других аспектов выполнения приложения. В качестве примера можно привести открытые соединения с базами данных, открытые файлы, потоки выполнения, графические объекты, а также такие объекты пользовательского интерфейса, как кисти, перья, элементы управления форм и формы как таковые. Все эти объекты не имеют непосредственного отношения к данным, с которыми фактически работает пользователь, но жизненно необходимы для фактического взаимодействия приложения с пользователем или с внешними источниками информации.

2. *Пользовательские данные*. К этой разновидности относятся фактические данные, в работе с которыми заинтересован пользователь. Под этими данными подразумевается та часть данных, которая удерживается в памяти, а не хранится в базе данных или файле на устройстве или вне его. Например, если мобильное приложение предназначено для того, чтобы облегчить пользователю проезд по улицам Лондона, то пользовательскими данными является информация о расположении улиц, которая в данный момент загружена в память. Если приложение предназначено для ведения инвентарного учета, то пользовательскими данными являются загруженные инвентаризационные данные. Если приложение представляет собой игру в шахматы, то пользовательскими данными является представление состояния шахматной доски в памяти машины.

2. Служебные данные приложения.

Эти данные представляют собой ресурсы, необходимые для эффективного функционирования приложения, а также отображения данных и манипулирования ими.

Рассмотрим в качестве примера простое приложение, ориентированное на работу с базами данных. Данное приложение обеспечивает хранение и обработку медицинских данных пациентов. Данные хранятся в базе данных и загружаются порциями, которые соответствуют отдельным пациентам, причем для загрузки или сохранения данных о пациенте требуется ввести защитный пароль. Для приложений такого рода можно выделить пять различных дискретных состояний, в которых пользователю для выполнения соответствующих задач предоставляются различные пользовательские интерфейсы. Такими состояниями являются следующими:

1. Загрузка данных из базы данных. Этот экран предоставляет пользователю возможность подтвердить свои права доступа к базе данных и загрузить данные конкретного пациента.
2. Сохранение данных в базе данных. Этот экран предоставляет пользователю возможность подтвердить свои права доступа к базе данных и сохранить данные конкретного пациента.
3. Основной экран приложения. Этот экран отображает данные истории болезни пациента, которые были загружены из базы данных, и предоставляет пользователю устройства возможность просматривать данные и переходить от одних данных к другим.
4. Экран, отображающий подробную информацию, необходимую для работы с конкретными данными и их редактирования. Этот экран отображается тогда, когда пользователю необходимо редактировать отдельные элементы данных о пациенте или вводить новые данные.
5. Экран диаграмм для отображения наборов точек, соответствующих данным. Этот экран предназначается для отображения графической информации, связанной с некоторыми аспектами истории болезни пациента. Например, могут быть построены диаграммы, представляющие изменение кровяного давления или количество белых кровяных телец с течением времени, по которым можно судить о наличии инфекции. К необходимым служебным данным приложения относятся следующие данные:
   - Графические перья и кисти, используемые для рисования диаграмм.
   - Объекты шрифтов, используемых для отображения надписей на диаграммах.
   - Кэшированные фоновые изображения.
6. Внеэкранная поверхность растрового изображения, используемая для подготовки изображения диаграммы перед его копированием на экран.

Некоторые из этих состояний разделяют общие ресурсы. Например, графическое перо черного цвета, или шрифт определенного размера, или растровое изображение могут использоваться в нескольких из перечисленных выше состояний. В соединении с базой данных нуждаются два состояния. Кроме того, некоторые объекты могут не требоваться для всех без исключения состояний, но их создание занимает длительное время; в этом случае целесообразно прибегнуть к кэшированию объектов, которое предварительно следует протестировать с точки зрения влияния на производительность.

Если объект имеет метод Dispose, то вы всегда можете вызвать этот метод, если необходимость в использовании данного объекта в приложении отпала. Метод Dispose () должен вызываться тогда, когда вы собираетесь удалить любые переменные ссылки на него, чтобы предоставить сборщику мусора возможность удалить этот объект из памяти. Тем самым гарантируется, что дорогостоящий объект, представляемый ресурсом, будет немедленно освобожден. Как и в других ситуациях, имеющих отношение к управлению памятью, если в случае приложений для настольных компьютеров такой подход является просто плодотворным, то в случае мобильных приложений, испытывающих дефицит системных ресурсов (таких, например, как дескрипторы операционной системы), его применение жизненно необходимо.

Управление объемом пользовательских данных, хранящихся в памяти.

Пользовательские данные представляют собой фактические данные, которые может просматривать или которыми может манипулировать пользователь приложения. Управление объемом и временем жизни пользовательских данных, хранящихся в памяти, может оказаться более сложным по сравнению с управлением служебными данными приложения, поскольку в зависимости от структуры и назначения приложения природа данных, сохраняемых в памяти, может быть самой различной. Пользовательские данные шахматной игры отличаются своей структурой от пользовательских данных истории болезни пациента. Состояние шахматной доски может храниться в целочисленном массиве фиксированных размеров. Объем данных истории болезни не может быть заранее определен; эти данные могут включать в себя результаты измерений, текстовые записи, изображения, ссылки на дополнительные данные и почти неограниченный объем любой другой подходящей информации.

Для управления потенциально сложными пользовательскими данными целесообразно использовать подход, предполагающий создание класса с хорошо продуманной инкапсуляцией, который и реализует управление состоянием, хранящимся в памяти в каждый момент времени. Этот класс отвечает за загрузку новых данных и освобождение памяти от старых данных, когда необходимость в них отпадает. Он управляет доступом к пользовательским данным извне и создает иллюзию неограниченных запасов памяти. Любой другой код приложения, осуществляющий доступ к данным извне инкапсулирующего класса, ничего не должен знать о внутреннем состоянии этого класса, реализующего фактическое управление пользовательскими данными. Возложение ответственности за управление всеми пользовательскими данными на определенный класс обеспечивает значительную гибкость в процессе проектирования. Некоторые из преимуществ такого подхода перечисляются ниже:

- Возможность автоматического управления объемом загруженных данных. Если вы обнаруживаете, что приложение испытывает острый дефицит памяти, можно уменьшить размеры окна данных, удерживаемых в памяти в каждый момент времени, не прибегая к внесению изменений в пределах всего приложения. Поскольку о том, какие данные кэшированы в памяти, а какие нуждаются в повторной загрузке, вне данного класса ничего не известно, вы получаете более гибкие возможности для настройки этого алгоритма.
- Возможность иметь различные реализации для различных классов устройств. Если ваши целевые устройства охватывают несколько различных классов, то вы имеете возможность настроить ограничения на использование памяти и накопителей для каждого из этих классов по отдельности. Мобильный телефон и устройство PDA могут иметь различные

характеристики памяти и различные возможности в отношении загрузки данных по требованию. Отмеченные различия могут потребовать от вас использования различных подходов к кэшированию данных. Концентрация соответствующей управляющей логики в одном месте существенно упрощает эту задачу.

Использование модели загрузки данных по требованию.

Для размещения объектов в памяти существуют две стратегии:

1. При вхождении приложения в новое состояние создаются все объекты, которые требуются для этого состояния. Достоинством этой стратегии является ее простота. Когда приложение переходит в новое состояние, вы просто вызываете функцию, которая и обеспечивает доступность и возможность использования всех необходимых объектов. Эта стратегия очень хорошо работает в тех случаях, когда имеется уверенность в том, что в ближайшее время приложению потребуются все созданные объекты. Возможные проблемы связаны с тем, что если ваше приложение находится в стадии становления и в его проект могут вноситься изменения то применение указанной стратегии может привести к хранению в памяти большого количества ненужных объектов. Поскольку старые объекты, необходимости в которых больше нет, все равно создаются и загружаются в память, то драгоценные ресурсы тратятся понапрасну. Будьте внимательны при групповом создании наборов объектов, ибо в процессе выполнения вашего приложения может наступить такой момент, когда создаваемые объекты не используются, но связанные с ними накладные расходы ухудшают производительность.

2. Создание любого объекта откладывается до тех пор, пока необходимость в его создании не станет очевидной. Эта модель немного сложнее в проектировании, но зато во многих случаях оказывается более эффективной, поскольку объекты создаются лишь тогда, когда в них возникает действительная необходимость. При обсуждении этой модели часто употребляются такие выражения, как "фабрика классов" ("class factory"), "диспетчер ресурсов" ("resource dispenser) и отложенная загрузка" ("lazy loading").

Приведенный ниже пример кода иллюстрирует два способа отложенного создания и кэширования глобально используемых графических ресурсов. Существует два способа создания объектов:

1. Пакетное создание групповых ресурсов. Приведенный ниже код создает списочный массив, содержащий четыре растровых изображения. Эти изображения являются кадрами анимации, поэтому они загружаются все вместе и помещаются в индексированный массив, откуда их можно легко извлекать. Программный код, которому требуется доступ к этой коллекции изображений, должен использовать вызов GraphicsGlobals. PlayerBitmapsCollection()/. Если массив изображений уже загружен в память, функция незамедлительно возвращает кэшированный объект. В противном случае отдельные ресурсы изображений сначала загружаются в массив и лишь затем возвращаются. Если приложение переходит в состояние, в котором пребывание изображений в памяти не требуются, код приложения может выполнить вызов GraphicsGlobals.g_PlayerBitmapsCollection_CleanUp();, в результате чего произойдет освобождение растровых ресурсов и массива. Системные ресурсы, задействованные для обслуживания растровых изображений, будут немедленно освобождены, а управляемая память, которую занимали эти объекты, будет соответствующим образом восстановлена в процессе сборки мусора.

2. Индивидуальное создание графических ресурсов. В случае ресурсов, которые не должны обязательно использоваться вместе, как в приведенном выше примере, часто оказывается удобным создать функцию кэшированного доступа, посредством которой и реализуется управление доступом к ресурсу. Когда происходит первое обращение к этой функции с запросом ресурса (например, GraphicsGlobals .g_GetBlackPen ();), она создает его экземпляр. В случае часто используемых ресурсов такой подход оказывается намного более эффективным, чем постоянное создание и уничтожение экземпляров ресурса всякий раз, когда он требуется для выполнения того или иного фрагмента кода. Создавая приведенный ниже код, я допустил, что все ресурсы должны освобождаться одновременно, и написал функцию (GraphicsGlobals.g_CleanUpDrawingResources ();), которая освобождает все каптированные ресурсы, которые были созданы. Эта функция должна вызываться тогда, когда приложение переходит в состояние, в котором эти ресурсы не требуются.

```
public static void g_PlayerBitmapsCollection_CleanUp(){
//Если не загружено ни одно изображение, то и память освобождать не от чего if(s_colPlayerBitmaps == null)
{ return; }
//Дать указание каждому из этих объектов освободить
//любые удерживаемые ими неуправляемые ресурсы s_Player_Bitmapl.Dispose();
s_Player_Bitmap2.Dispose();
s_Player_Bitmap3.Dispose();
s_Player_Bitmap4.Dispose();
//Обнулить каждую из этих переменных, чтобы им не соответствовали
//никакие объекты в памяти
s_Player_Bitmapl = null;
s_Player_Bitmap2 = null;
s_Player_Bitmap3 = null;
s_Player_Bitmap4 = null;
//Избавиться от массива s_colPlayerBitmaps = null;
//Функция: возвращает коллекцию изображений
public static System.Collections.ArrayList g_PlayerBitmapsCollection()
{
//
//Если изображения уже загружены, их достаточно только возвратить
//
if(s_colPlayerBitmaps != null)   {return s_colPlayerBitmaps;)
  //Загрузить изображения как ресурсы из исполняемого двоичного файла
System.Reflection.Assembly thisAssembly =System.Reflection.Assembly.GetExecutingAssembly();
```

```
System.Reflection.AssemblyName thisAssemblyName =thisAssembly.GetName();
string assemblyName = thisAssemblyName.Name; /
/Загрузить изображения
s_Player_Bitmapl = new System.Drawing.Bitmap(thisAssembly.GetManifestResourceStream(assemblyName +
".HankJRightRunl.bmp"));
s_Player_Bitmap2 = new System.Drawing.Bitmap(thisAssembly.GetManifestResourceStream(assemblyName +
".Hank_RightRun2.bmp"));
s_Player_Bitmap3 = new System.Drawing.Bitmap(thisAssembly.GetManifestResourceStream(assemblyName +
".Hank_LeftRunl.bmp"));
s_Player_Bitmap4 = new System.Drawing.Bitmap(thisAssembly.GetManifestResourceStream(assemblyName +
".Hank_LeftRun2.bmp"));
//Добавить изображения в коллекцию
s_colPlayerBitmaps = new System.Collections.ArrayList();
s_colPlayerBitmaps.Add(s_Player_Bitmapl);
  s_colPlayerBitmaps.Add(s_Player_Bitmap2);
s_colPlayerBitmaps.Add(s_Player_Bitmap3);
s_colPlayerBitmaps.Add(s_PlayerJBitmap4);
//Возвратить коллекцию return s_colPlayerBitmaps;
}
private static System.Drawing.Pen s_blackPen;
private static System.Drawing.Pen s_whitePen;
private static System.Drawing.Imaging.ImageAttributes s_ImageAttribute;
private static System.Drawing.Font s_boldFont;
//
//Вызывается для освобождения от любых графических
//ресурсов, которые могли быть кэшированы
//
private static void g_CleanUpDrawingResources() {
//Освободить память от черного пера, если таковое имеется if(s_blackPen != null)
{s_blackPen.Dispose(); s_blackPen = null;}
// Освободить память от белого пера, если таковое имеется if(s_whitePen != null)
{ s_whitePen.Dispose(); s_whitePen = null;}
//Освободить память от атрибута ImageAttribute, если таковой имеется.
//Примечание. Метод Dispose() для этого типа не предусмотрен,
//поскольку все его данные являются управляемыми
if(s_ImageAttribute != null) {s_ImageAttribute = null;}
//Освободить память от полужирного шрифта, если таковой имеется
if(s_boldFont != null)
{s_boldFont.Dispose(); s_boldFont = null;}
//
//Эта функция позволяет получить доступ
//к черному перу, находящемуся в кэш-памяти
//
private static System.Drawing.Pen g_GetBlackPen() {
//Если перо еще не существует, создать его
if(s_blackPen == null)
{
s_blackPen = new System.Drawing.Pen( System.Drawing.Color.Black) ;
}
//Возвратить черное перо return s blackPen;
//
//Эта функция позволяет получить доступ
//к белому перу, находящемуся в кэш-памяти
//
private static System.Drawing.Pen g_GetWhitePen() {
//Если перо еще не существует, создать его
if(s_whitePen == null)
{s_whitePen     = new System.Drawing.Pen(
System.Drawing.Color.White);} //Возвратить белое перо return s_whitePen;
}
//
//Эта функция позволяет получить доступ
//к полужирному шрифту, находящемуся в кэш-памяти
//
private static System.Drawing.Font g_GetBoldFont() {
//Если перо еще не существует, создать его
if(s_boldFont == null)
{
```

```
        s_boldFont     = new System.Drawing.Font( System.Drawing.FontFamily.GenericSerif, 10, System.Drawing.FontStyle.Bold)
;
        }
        //Возвратить полужирный шрифт return s_boldFont;
        }
        //
        //Эта функция позволяет осуществлять доступ
        //к находящемуся в кэш-памяти объекту imageAttributes,
        // который мы используем для изображений с прозрачностью
        //  private static System.Drawing.Imaging.ImageAttributes g_GetTransparencyImageAttribute()
        {
        //Если объект не существует, создать его
        if(s_ImageAttribute == null)
        {
        //Создать атрибут изображения s_ImageAttribute =new System.Drawing.Imaging.ImageAttributes();
        s_ImageAttribute.SetColorKey(System.Drawing.Color.White,
        System.Drawing.Color.White);
        }
        //Возвратить его return s_ImageAttribute;
        }
        } //Конец класса
```

Лабораторная работа № 4. Анализ предметной области. Выявление функциональных требований к приложению.

Управление памятью на микроскопическом "уровне алгоритма".

Современные языки программирования, библиотеки классов и управляемые среды времени выполнения позволили значительно повысить продуктивность написания программ. В то же время, избавляя программиста от необходимости задумываться о низкоуровневом распределении памяти, в котором нуждаются алгоритмы, они невольно создают предпосылки для написания неэффективного кода. Неэффективность кода может быть обусловлена причинами двоякого рода:

1. Вычислительная неэффективность алгоритма. Этот вид неэффективности наблюдается в тех случаях, когда спроектированный вами алгоритм предусматривает интенсивные вычисления или выполнение большего количества циклов, чем это объективно необходимо, от чего можно было бы избавиться, используя более эффективные алгоритмы. В качестве классического примера можно привести сортировку массива данных. Иногда у вас может появляться возможность выбирать между несколькими возможными вариантами алгоритмов сортировки, отдельными частными случаями которых могут, например, быть алгоритмы "порядка N" (линейная зависимость времени вычислений от количества сортируемых элементов), "порядка N*Log(N)" (зависимость времени вычислений от количества сортируемых элементов отличается от линейной, но остается все же лучшей, чем экспоненциальная) или "порядка N^A2 " (экспоненциальная зависимость времени вычислений от количества сортируемых элементов). Кроме вышеперечисленных "порядков" возможно множество других (например, N^A3). Выбор наиболее подходящего алгоритма зависит от объема данных, с которыми вы работаете, объема доступной памяти и ряда других факторов, например, от состояния рабочих данных. Отдельные стратегии, например, предварительная обработка данных перед отправкой их на устройство или хранение данных в формате, специфическом для использования памяти в качестве хранилища, способны обеспечить значительное повышение производительности алгоритма. Существует огромное количество компьютерной литературы, посвященной проектированию эффективных алгоритмов и оценке их быстродействия, поэтому никаких попыток более подробного анализа этих вопросов в данной книге не делается. Необходимо только отметить, что чем больше объем обрабатываемых данных, тем ответственнее необходимо отнестись к принятию решения относительно выбора вычислительного алгоритма. Во всех затруднительных случаях тщательно анализируйте алгоритм и обращайтесь к существующей литературе по этому вопросу. Очень часто оказывается так, что кто-то другой уже прошел этот путь, и вам остается лишь перенять их опыт.

2. Неэффективное распределение памяти. После того как вы определитесь со стратегией алгоритма, следующим фактором, от которого в значительной степени зависит производительность приложения, является способ реализации этого алгоритма. При этом едва ли не наибольшие усилия вы должны приложить к тому, чтобы избежать распределения лишних объемов памяти, особенно если память распределяется в циклах. В данном разделе этого курса основное внимание уделяется именно этому вопросу.

Вашей целью должно быть распределение "нулевых объемов памяти" внутри циклов в написанном вами коде. Существуют случаи, когда это является неизбежным, как, например, при построении дерева объектов, которое требует размещения в памяти новых узлов для помещения их в иерархическую структуру. Во многих других случаях эффективность приложения можно существенно повысить, тщательно анализируя распределение памяти для каждого объекта и рассматривая альтернативные решения. Чего, как правило, следует избегать - так это выполнения операций размещения объектов в памяти и удаления их из памяти внутри алгоритмических циклов.

4. "Структуры" и .NET Compact Framework.

Во многих случаях, если вы хотите инкапсулировать некоторые простые данные, то для локальных переменных внутри функций гораздо эффективнее использовать не объекты, а структуры. Структура - это просто удобный способ сгруппировать в одном пакете взаимосвязанные данные, а не передавать их в виде отдельных переменных.

Структуры обладают более простыми свойствами по сравнению с объектами, но могут "упаковываться" в объекты и передаваться внутри программы так же, как они, если в этом возникает необходимость. Использование структур предоставляет определенные удобства и может привести к некоторому увеличению производительности (по сравнению с вариантом, когда используются объекты), но поскольку они выглядят, а во многих случаях и действуют подобно объектам и

могут заключаться в объекты-оболочки, необходимо тщательно взвешивать, когда их следует использовать, чтобы избежать дополнительных накладных расходов и не создать лишнего мусора. В сомнительных случаях тестируйте алгоритмы, используя как отдельные переменные (например, базовые типы, подобные int, string, double), так и структуры, чтобы сравнить производительность приложения в обоих случаях и убедиться в том, что она остается примерно одинаковой.

5. Использование строк в алгоритмах.

Современные языки программирования позволяют очень легко работать со строками, создавать их, разбивать, копировать и объединять. Рассмотрим, например, следующие простые операторы:

```
string strl = "internet";
string str2 = "explorer";
string str3 = strl + str2;
string str3 = str3 + str3;
```

Простота операций со строками ведет к их нерациональному использованию. Не оптимизированная обработка строк является одной из наиболее вероятных причин плохой производительности. Ниже представлены некоторые рекомендации и правила, которыми следует руководствоваться при работе со строками.

1. Строки неизменчивы (постоянны). Этот странный термин неизменчивый (immutable) просто означает, что текстовые данные строки не могут быть изменены в памяти. Те операции в коде, которые, как вам кажется, изменяют данные строки, на самом деле создают новую строку. Постоянство обладает некоторыми весьма привлекательными свойствами. Например, поскольку строковые данные сами по себе являются статическими, несколько переменных могут указывать на одни и те же данные; благодаря этому присвоение одной строковой переменной значения другой сводится к простому копированию "указателя" вместо глубокого копирования всех данных, которые ему соответствуют. Отрицательной стороной неизменчивости является невозможность изменения данных. Если вы хотите изменить, добавить или отсечь данные, то эти изменения будут отражаться в новой копии строки.

2. Когда па строковые данные не ссылается пи одна "активная" ("live") переменная, они становятся "мусором". Рассмотрим пример:

```
string     strl = "internet";
  //"internet" - статические данные, скомпилированные
//в двоичные данные вашего приложения
string str2 = strl + strl;
//только что была создана новая строка, являющаяся результатом конкатенации двух строк
str2 = "explorer";
//Поскольку отсутствуют другие переменные, указывающие на те данные, на которые указывала переменная str2,
//эти данные становятся мусором, и память должна быть
//очищена от них.
```

Если вы хотите сослаться на некоторую часть строки, то во многих случаях это проще всего сделать, используя целочисленные индексы в строке. Поскольку строки - это данные, представленные массивами символов, то использование индексов для получения этих данных не составляет труда. Существует множество функций, позволяющих осуществлять поиск и просмотр данных внутри строк (но только не изменять эти данные!).

Если вы создаете новые строки внутри циклов, настоятельно рекомендуется рассмотреть возможность использования объекта StringBuilder. Все виды строк создаются на основе других переменных, обрабатываемых в циклах. Типичным примером динамического создания строк может служить цикл, генерирующий текстовый отчет, каждая строка которого содержит следующие данные:

```
//Неэффективный код, выполняющийся внутри цикла
{
myString = myString +"CustomerID: "
+ System.Convert.ToString(customer[idx].id) + ", Name: " + System.Convert.ToString(customer[idx].name);
      }
```

Вместо того чтобы конкатенировать строки и создавать новую строку, для создания отчета можно было бы использовать класс StringBuilder. Класс StringBuilder очень удобно использовать для работы с массивами переменной размерности с целью создания строк. Он позволяет эффективно изменять длину или содержимое массива и, что самое важное, создавать новые строки на ос нове символьных массивов. Обязательно изучите класс StringBuilder, поскольку умение использовать его имеет решающее значение для написания эффективных алгоритмов, генерирующих строковые данные.

3. Измеряйте объективные количественные показатели своих алгоритмов. Занимаясь написанием алгоритма обработки строк, тестируйте его быстродействие! Испробуйте несколько различных подходов. Вы очень быстро научитесь распознавать, какой алгоритм будет эффективным, а какой - нет.

В листинге представлены два аналогичных алгоритма, которые приводят к одному и тому же результату. В обоих алгоритмах осуществляется инкрементирование счетчика, и каждый раз, когда значение счетчика увеличивается, его строковое представление добавляется в расширяемый фрагмент текста. Оба алгоритма выполняют одинаковое количество итераций и характеризуются одинаковой степенью сложности написания, тем не менее, один из них работает гораздо быстрее другого.

```
//Для имитации создания типичного набора строк используются
//обычные строки
private void buttonl_Click(object sender, System.EventArgs e)
{
//Вызвать сборщик мусора, чтобы тест
//начинался с чистого состояния.
System.GC.Collect ();
int numberToStore = 0;
PerformanceSampling.StartSample (0, "StringAllocaitons");
```

```
string total_result = "";
for (int outer_loop = 0; outer_loop < LOOP_ITERATIONS; outer_loop++)
{
//Сбросить старый результат
total_result = "";
//Выполнять цикл до максимального значения x_counter, каждый
//раз присоединяя очередную тестовую строку к рабочей строке
for(int x_counter = 0; x_counter < COUNT_UNTIL; x_counter++)
{
total_result = total_result + numberToStore.ToString();
//Увеличить значение счетчика
numberToStore ++;
}
}
PerformanceSampling.StopSample(0);
//Отобразить длину строки
System.Windows.Forms.MessageBox.Show("Длина строки: " +total_result.Length.ToString());
//Отобразить строку
System.Windows.Forms.MessageBox.Show("Строка : " +total_result);
//Отобразить длительность интервала времени, ушедшего на вычисления
System.Windows.Forms.MessageBox.Show(PerformanceSampling.GetSampleDurationText(0));
}
//Для имитации создания типичного набора строк используется
//объект StringBuilder
private void button2_Click(object sender, System.EventArgs e) {
//Вызвать сборщик мусора, чтобы тест
//начинался с чистого состояния.
System.GC.Collect();
System.Text.StringBuilder sb = new System.Text.StringBuilder(); string total_result = ""; int numberToStore = 0;
PerformanceSampling.StartSample(1, "StringBuilder");
for (int outerJLoop = 0; outer_loop < LOOP_ITERATIONS; outer_loop++)
{
//Очистить объект StringBuilder (не создавая нового объекта)
sb.Length = 0;
//Очистить строку со старым результатом
total_result = "";
//Выполнять цикл до максимального значения x_counter, каждый раз
//присоединяя очередную тестовую
  //строку к рабочей строке
for(int x_counter = 0; x_counter < COUNTJJNTIL; x_counter++)
{
sb.Append(numberToStore );
sb.Append(", ");
//Увеличить значение счетчика numberToStore ++;
}
//Имитируем выполнение некоторых операций над строкой...
total_result = sb.ToStringO ;
}
PerformanceSampling.StopSample(1);
//Отобразить длину строки
System.Windows.Forms.MessageBox.Show("Длина строки: "+ total_result.Length.ToString()) ;
//Отобразить строку
System.Windows.Forms.MessageBox.Show("String : " + total_result);
//Отобразить длительность интервала времени, ушедшего на вычисления System.Windows.Forms.MessageBox.Show(
PerformanceSampling.GetSampleDurationText(1));
}
```

Лабораторная работа № 5. Этапы проектирования приложения для мобильного устройства.

Целью данной лабораторной работы является знакомство с языком *XML* и возможностями его применения в разработке мобильных приложений.

Задание на лабораторную работу.

1. Изучить теоретические основы работы с XML - данными
2. Реализовать алгоритм записи и чтения данных их XML - документа, основанный на модели XML DOM.
3. Реализовать алгоритм записи и чтения данных их XML - документа, основанный на однонаправленном чтении - записи данных.

Язык разметки документов - это набор специальных инструкций, называемых тэгами, предназначенных для формирования в документах какой-либо структуры и определения отношений между различными элементами этой структуры. Тэги языка, или, как их иногда называют, *управляющие* дескрипторы, в таких документах каким-то образом кодируются, выделяются относительно основного содержимого документа и служат в качестве инструкций для программы, производящей

показ содержимого документа на стороне клиента. В самых первых системах для обозначения этих команд использовались символы "<" и ">", внутри которых помещались названия инструкций и их параметры. Сейчас такой способ обозначения тэгов является стандартным.

Использование гипертекстовой разбивки текстового документа в современных информационных системах во многом связано с тем, что *гипертекст* позволяет создавать механизм нелинейного просмотра информации. В таких системах данные представляются не в виде непрерывного потока текстовой информации, а набором взаимосвязанных компонентов, переход по которым осуществляется при помощи гиперссылок.

XML (*Extensible Markup Language*) - это *язык разметки*, описывающий *целый класс* объектов данных, называемых *XML*- документами. Этот язык используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов. Т.е. сам по себе *XML* не содержит никаких тэгов, предназначенных для разметки, он просто определяет порядок их создания.

При всех тех возможностях, которые кроются в *XML*, легко предположить, что этот язык должен был бы привлекаться всегда, когда требуется осуществлять *обмен данными* или их хранение. Тем не менее, это не так. Гибкость *XML* придает ему мощь, но это дается за счет увеличения размера документов. Если требуется осуществить *обмен данными* сравнительно небольшого объема, то *XML* великолепно подходит для этой цели Так, при загрузке 20 строк форматированных записей *базы данных* размер *XML*-файла может достигать 20 Кбайт, тогда как при использовании для передачи данных двоичного формата они могут быть сжаты до 2 Кбайт или менее.

Достоинства XML.

- XML - язык разметки, позволяющий отобразить двоичные данные в текст, читаемый человеком и анализируемый компьютером;
- XML поддерживает Юникод;
- в формате XML могут быть описаны такие структуры данных как записи, списки и деревья;
- XML - это самодокументируемый формат, который описывает структуру и имена полей так же как и значения полей;
- XML имеет строго определённый синтаксис и требования к анализу, что позволяет ему оставаться простым, эффективным и непротиворечивым. Одновременно с этим, разные разработчики не ограничены в выборе экспрессивных методов (например, можно моделировать данные, помещая значения в параметры тегов или в тело тегов, можно использовать различные языки и нотации для именования тегов и т. д.);
- XML - формат, основанный на международных стандартах;
- Иерархическая структура XML подходит для описания практически любых типов документов, кроме аудио и видео мультимедийных потоков, растровых изображений, сетевых структур данных и двоичных данных;
- XML представляет собой простой текст, свободный от лицензирования и каких-либо ограничений;
- XML не зависит от платформы;
- XML является подмножеством SGML (который используется с 1986 года). Уже накоплен большой опыт работы с языком и созданы специализированные приложения;
- XML не накладывает требований на расположение символов в строке;[http://ru.wikipedia.org/wiki/XML#cite_note-json-1]
- В отличие от бинарных форматов, XML содержит метаданные об именах, типах и классах описываемых объектов, по которым приложение может обработать документ неизвестной структуры (например, для динамического построения интерфейсов[http://ru.wikipedia.org/wiki/XML#cite_note-2]);
- XML имеет реализации парсеров для всех современных языков программирования;[http://ru.wikipedia.org/wiki/XML#cite_note-3]
- XML поддерживается на низком аппаратном, микропрограммном и программном уровнях в современных аппаратных решениях.[http://ru.wikipedia.org/wiki/XML#cite_note-4]

Недостатки XML.

- Синтаксис XML избыточен.
- Размер XML документа существенно больше бинарного представления тех же данных. В грубых оценках величину этого фактора принимают за 1 порядок (в 10 раз).
- Размер XML документа существенно больше, чем документа в альтернативных текстовых форматах передачи данных (например JSON, YAML) и особенно в форматах данных, оптимизированных для конкретного случая использования.
- Избыточность XML может повлиять на эффективность приложения. Возрастает стоимость хранения, обработки и передачи данных.
- XML содержит мета-данные (об именах полей, классов, вложенности структур), и одновременно XML позиционируется как язык взаимодействия открытых систем. При передаче между системами большого количества объектов одного типа (одной структуры), передавать метаданные повторно нет смысла, хотя они содержатся в каждом экземпляре XML описания.
- Для большого количества задач не нужна вся мощь синтаксиса XML и можно использовать значительно более простые и производительные решения.
- Неоднозначность моделирования.
- Нет общепринятой методологии для моделирования данных в XML, в то время как для реляционной модели и объектно-ориентированной такие средства разработаны и базируются на реляционной алгебре, системном подходе и системном анализе.
- В природе есть множество объектов и явлений, для описания которых разные структуры данных (сетевая, реляционная, иерархическая) являются естественными, и отображение объекта в неестественную для него модель является болезненным для его сути. В случае с реляционной и иерархической моделями определены процедуры декомпозиции, обеспечивающие относительную однозначность, чего нельзя сказать о сетевой модели.

- В результате большой гибкости языка и отсутствия строгих ограничений, одна и та же структура может быть представлена множеством способов (различными разработчиками), например, значение может быть записано как атрибут тега или как тело тега и т. д.
- Поддержка многих языков в именовании тегов дает возможность назвать, например вес русским словом, в таком случае компьютер никак не сможет установить соответствия этого поля с полем weight в англоязычной версии программы и с полями в версиях модели объекта на множестве других языков.
- XML не содержит встроенной в язык поддержки типов данных. В нём нет *строгой типизации*, то есть понятий "целых чисел", "строк", "дат", "булевых значений" и т. д.
- *Иерархическая модель данных*, предлагаемая XML, ограничена по сравнению с реляционной моделью и объектно-ориентированными графами и *сетевой моделью данных*.

Сохранение данных в виде XML.

Как и в случае HTML, в XML данные сохраняются в виде текста, заключенного между дескрипторами, которые дополняют передаваемые данные контекстом:

<UserInfo>
<UserID> 12 </UserID> <UserName> Bob </UserName>
<UserAddress> Someplace, Somewhere </UserName> </UserInfo>

Следует отметить, что в XML те же данные можно сохранить с использованием атрибутов, например:

<UserInfo UserID="12" UserName="Bob" UserAddress="Someplace, Somewhere"> </UserInfo>

При необходимости можно использовать сочетания дескрипторов и атрибутов. Какой формат XML окажется самым подходящим, зависит от конкретной ситуации. Атрибуты проще использовать, но дескрипторы обладают большей гибкостью, поскольку они могут иметь атрибуты и вложенные дескрипторы.

Иерархическая структура XML-данных.

Очень важно хорошо понимать иерархическую структуру *XML*. Представляйте себе *XML*-документ в виде дерева объектов, у каждого из которых могут иметься дополнительные дочерние объекты. Для демонстрации этого изменим приведенный выше пример таким образом, чтобы внести в него дополнительную иерархию:

<UserInfo>
<UserID> 12 </UserID> <Name>
<FirstName> Иво </FirstName> <LastName> Салмре </LastName> </Name> <Address>
<Street>10 НекаяУлица</5сгеег.> <City>Cn3^</City> <State>WA</State> </Address> </UserInfo>

Здесь мы сделали узлы Name и Address подузлами Userlnfo.

Что касается практического использования, то в большинстве случаем *XML*-данные, участвующие в обмене, попадают в промежуток между двумя крайними категориями: строго структурированными данными и данными свободной формы. Какой уровень строгости следует применять к форматам данных, решают совместно как отправитель, так и получатель данных. Вычислительный узел, генерирующий *XML*-документы, может придерживаться строго определенной схемы или же просто может выбирать ту форму *XML*-кодирования, которая является для него наиболее удобной. Аналогичным образом, вычислительный узел, получающий *XML*-документы, может либо выполнять проверку их содержимого на предмет соответствия предполагаемой схеме, либо предполагать, что данные форматированы корректно, и сразу же пытаться осуществить синтаксический *анализ* результатов. Поскольку такие *операции*, как *верификация* схемы, могут быть трудоемкими в вычислительном отношении, то в тех случаях, когда такая проверка нужна, лучше, чтобы она осуществлялась на сервере до передачи данных устройству.

При работе с XML-данными вы можете использовать один из трех основных подходов:

1. Создать собственный "оптимизированный" анализатор с нуля. При наличии ранее разработанных и протестированных методик это почти никогда не стоит делать. Причина, по которой применять такой подход категорически не рекомендуется, заключается в том, что получаемые при этом преимущества лишь в редких случаях окупают усилия, затрачиваемые на разработку и последующее сопровождение соответствующих программ.
2. Использовать высокоуровневые универсальные методы синтаксического анализа с произвольным, доступом, основанные на модели XML DOM. DOM (Document Object Model- объектная модель документов) обеспечивает возможность работы с XML-данными, хранящимися в памяти в виде иерархического дерева объектов. В результате использования высокоуровневого API-интерфейса для работы с XML вы получаете в высшей степени надежный код, удобный в сопровождении. Такой подход является оптимальным для небольших XML-документов, а также документов, при работе с которыми требуется постоянный произвольный доступ ко всем частям дерева XML-документа, или документов, которые должны быть заново целиком сохранены в файле на диске.
3. Использовать низкоуровневый API-интерфейс XML, обеспечивающий выполнение лишь однонаправленных операций чтения-записи данных. Применение низкоуровневых API-интерфейсов позволяет максимально повысить производительность, но возлагает дополнительную нагрузку на программистов. Эти API-интерфейсы поддерживают выполнение операций чтения-записи данных только в прямом направлении и позволяют считывать или записывать данные XML-дерева в виде потока XML-элементов без сохранения всего документа в памяти. В случае мобильных устройств, для которых память всегда является дефицитным ресурсом, и особенно при работе с большими объемами данных или данными, предназначенными только для чтения, только такой подход и обеспечивает достижение приемлемой производительности. Он представляет собой хорошую основу, являющуюся промежуточной между использованием высокоуровневых АР1-интер(рейсов и развертыванием собственной методики. Такой путь является разумным, если привлечение высокоуровневых API-интерфейсов для удовлетворения ваших нужд требует интенсивных дополнительных вычислений и приводит к чрезмерному расходу памяти.

Более подробно рассмотрим второй и третий варианты работы с *XML* - данными

XML DOM

DOM (от англ. *Document Object Model* - "объектная модель документа") - это независящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому документов, а также изменять содержимое, структуру и оформление документов.

Модель DOM не накладывает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями родительский-дочерний.

Изначально различные браузеры имели собственные модели документов (DOM), не совместимые с остальными. Для того, чтобы обеспечить взаимную и обратную совместимость, специалисты международного консорциума W3C классифицировали эту модель по уровням, для каждого из которых была создана своя спецификация. Все эти спецификации объединены в общую группу, носящую название W3C DOM.

Объектная модель XML-документа, или DOM (Document Object Model) состоит из группы программных объектов, представляющих различные компоненты XML-документа. Свойства и методы этих объектов позволяют использовать сценарии для отображения XML-документа с HTML-страницы. DOM хранит данные в древообразной структуре, отражающей иерархическую структуру XML-документа, и предоставляет доступ к любым компонентам XML-документа, включая элементы, атрибуты, инструкции по обработке, комментарии и объявления нотаций и примитивов.

Чтобы получить доступ к XML-документу с использованием DOM, вы должны связать XML-документ с HTML-страницей. На HTML-странице это можно сделать через так называемый фрагмент данных:

<XML ID="dsoDoc" SRC="Sample.xml"></XML>

Идентификатор ID, который вы назначаете фрагменту данных, указывает на *DSO* (Data Source Object) документа. Свойство объекта *DSO* "XMLDocument" содержит корневой объект DOM, называемый также "узел документа". Вы можете использовать это свойство для доступа к DOM в сценариях:

xmlDOM_Document = dsoDoc.XMLDocument;

Полный перечень свойств, методов и событий объекта *DSO* документа вы можете получить в MSDN, воспользовавшись, например, поиском по словосочетанию "xml Object". Если вы хотите иметь доступ к нескольким XML-документам с HTML-страницы, вы можете поместить фрагмент данных для каждого из них. Кроме того, вы можете включить несколько фрагментов данных для одного XML-документа. Последний приём может оказаться полезным для поддержки нескольких различных версий данных XML, если ваша страница модифицирует эти данные.

XML DOM работает с представлением данных в виде дерева объектов в памяти. Каждый XML-элемент представляется хранящимся в памяти объектом. Можно считать, что подход XML DOM в значительной степени основан на концепции состояния в том смысле, что все данные, необходимые для воссоздания XML-документа после его считывания, загружаются как состояние. XML-деревья могут создаваться в памяти, а затем сериализовываться в файлы или через сетевые потоки. Аналогичным образом, любое XML-содержимое или XML-документ, полученный с использованием любого потока данных, могут быть использованы для *заполнения дерева* XML DOM в памяти.

Создание дерева объектов в памяти является удобным способом работы с данными среднего объема, которые требуется всего лишь последовательно обновлять. XML-файл размером 20 Кбайт можно довольно быстро загрузить в память и после работы с ним в памяти как с деревом объектов вновь сохранить в файловой системе. Коль скоро объемы интересующих нас данных невелики, модель XML DOM предоставляет отличные возможности для создания XML-документов, обработки их в памяти и вывода XML-данных в файл или сетевой поток.

Применимость DOM-подхода ограничивается как объемом памяти, доступной для размещения сгенерированного анализатором дерева объектов, так и наличием резер вов вычислительной мощности, необходимых для разбора всего дерева XML-данных. Недостатком подхода, основанного на XML DOM, является его монолитность; прежде чем вы сможете получить доступ к XML-данным, вы должны выполнить синтаксический анализ документа и разместить в памяти все содержимое файла или потока. Если вам необходимо работать лишь с небольшим объемом данных, содержащихся в файле большого размера, то доступ к этим данным будет сопряжен с большими накладными расходами.

```
using System;
//
//Демонстрирует сохранение и загрузку файлов с
//использованием объектной модели документов ХМЪ
//
public class SaveAndLoadXML_UseDOM {
//XML-дескрипторы, которые мы будем использовать в нашем документе
const string XML_R00T_TAG = "AllMyData";
const string XML_USERINFO_TAG = "Userlnfo";
const string XML_USERID_TAG = "UserlD";
const string XML_NAMEINF0_TAG = "Name";
const string XML_FIRSTNAME_TAG = "FirstName";
const string XML_LASTNAME_TAG = "LastName";
//
//Загружает пользовательское состояние //
// [in] fileName:       Имя файла, используемого для сохранения данных
// [out] userld:        Загруженный идентификатор пользователя
// [out] firstName: Загруженное имя пользователя
// [out] lastName:    Загруженная фамилия пользователя
//
public static void XML_LoadUserInfo(string fileName,
out int userld, out string firstName, out string lastName)
{
//Начинаем с нулевых значений userld = 0; firstName = ""; lastName = ,,n;
//Предполагаем, что данные еще не загружены bool gotUserlnfoData = false; System.Xml.XmlDocument XmlDocument =
new System.Xml.XmlDocument();
xmlDocument.Load(fileName);
```

```
//Получить корневой узел
System.Xml.XmlElement rootElement;
rootElement =(System.Xml.XmlElement)xmlDocument.ChildNodes[0];
//Убедиться в том, что корневой узел согласуется с ожидаемым текстом,
//ибо противное означает, что мы имеем дело с каким-то другим XML-файлом if (rootElement.Name != XML_R00T_TAG) {
throw new Exception("Тип корневого узла не совпадает с ожидаемым!");
}
//
//Простой конечный автомат для итеративного обхода всех узлов
//
foreach(System.Xml.XmlElement childOf_RootNode in rootElement.ChildNodes)
{
//Если это узел Userlnfo, то мы хотим просмотреть его содержимое
if(childOf_RootNode.Name = XML_USERINFO_TAG)
I
gotUserlnfoData     = true; //Пользовательские данные найдены
//
//Загрузить каждый из подэлементов
//
foreach(System.Xml.XmlElement child_UserDataNode in childOf_RootNode.ChildNodes)
{
//Идентификатор пользователя (UserlD)
if(child_UserDataNode.Name = XML_USERID_TAG)
{
userld = System.Convert.ToInt32( child_UserDataNode.InnerText);
}
//ФИО пользователя (UserName) else if(child UserDataNode.Name == XML NAMEINFO TAG)
foreach(System.Xml.XmlElement child_Name in childJJserDataNode.ChildNodes)
{
//Имя (FirstName)
if(child_Name.Name == XML_FIRSTNAME_TAG) {
firstName = child_Name.InnerText;
}
//Фамилия (LastName) else if(child_Name.Name == XML_LASTNAME_TAG) {
lastName = child_Name.InnerText;
}
) //Конец цикла разбора UserName ) //Конец оператора if, осуществляющего проверку UserName }
//Конец цикла разбора Userlnfo ) Конец оператора if, осуществляющего проверку Userlnfo }
//Конец цикла разбора корневого узла
if (gotUserlnfoData == false)
{
throw new Exception("Данные пользователя в XML-документе не найдены!");
}
//
//Сохраняет пользовательское состояние //
// [in] fileName:     Имя файла, используемого для сохранения данных
// [in] userld:          Идентификатор пользователя, который мы хотим сохранить
// [in] firstName: Имя пользователя, которое мы хотим сохранить
// [in] lastName:     Фамилия пользователя, которую мы хотим сохранить
//
public static void XML_SaveUserInfo(string fileName, int userld, string firstName, string lastName)
{
System.Xml.XmlDocument xmlDocument = new System.Xml.XmlDocument();
//
//Добавить элемент документа высшего уровня
//
System.Xml.XmlElement rootNodeForDocument; rootNodeForDocument = xmlDocument.CreateElement(
XML_R00T_TAG); xmlDocumen t.Appen dCh i1d(rootNode For Document);
//
//Добавить данные в элемент Userlnfo
//
System.Xml.XmlElement topNodeForUserData; topNodeForUserData = xmlDocument.CreateElement(
XML_USERINFO_TAG); rootNodeForDocument.AppendChild(topNodeForUserData);
//
//Добавить значение UserlD в наш документ
//
```

```
//Создать подузел для информации о пространстве имен System.Xml.XmlElement subNodeForUserlD; subNodeForUserlD =
xmlDocument.CreateElement(XML_USERID_TAG); subNodeForUserlD.InnerText =
System.Convert.ToString(userld);
//Присоединить подузел UserlD к узлу высшего уровня topNodeForUserData.AppendChild(subNodeForUserlD);
//
//Добавить все значения Namelnfo в наш документ
//
//Создать подузел для информации о пространстве имен System.Xml.XmlElement subNodeForNamelnfo;
subNodeForNamelnfo = xmlDocument.CreateElement( XML_NAMEINFO_TAG);
//Имя (FirstName)
System.Xml.XmlElement subNodeFirstName; subNodeFirstName = xmlDocument.CreateElement(
XML_FIRS TNAME_TAG); subNodeFirstName.InnerText = firstName;
//Фамилия (LastName)
System.Xml.XmlElement subNodeLastName; subNodeLastName = xmlDocument.CreateElement(
XML_LASTNAME_TAG); subNodeLastName.InnerText = lastName;
//Присоединить подузлы имени и фамилии к родительскому //узлу Namelnfo
subNodeForNamelnfo.AppendChild(subNodeFirstName); subNodeForNamelnfo.AppendChild(subNodeLastName);
//Присоединить подузел Namelnfo (вместе с его дочерними узлами) //к узлу высшего уровня
topNodeForUserData.AppendChild(subNodeForNamelnfo);
//
//Сохранить документ
//
try
{
xmlDocument.Save(fileName);
)
catch (System.Exception ex) {
System.Windows.Forms.MessageBox.Show(
"Ошибка при сохранении XML-документа - " + ex.Message);
}
} //Конец функции } //Конец класса
private void buttonl_Click(object sender, System.EventArgs e)
{
const string FILENAME = "TestFileName.XML";
//Сохранить, используя XML DOM
SaveAndLoadXML_UseDOM.XML_SaveUserInfo(FILENAME, 14, "Ivo","Salmre");
//Сохранить, используя объект однонаправленной записи XMLWriter
//SaveAndLoadXML_UseReaderWriter.XML_SaveUserInfo(FILENAME,
//  18, "Ivo", "Salmre");
int userlD; string firstName; string lastName;
//Загрузить, используя XML DOM
SaveAndLoadXML_UseDOM.XML_LoadUserInfo(FILENAME, out userlD,
out firstName, out lastName);
//Загрузить, используя объект однонаправленного чтения XMLReader
//SaveAndLoadXML_UseReaderWriter.XML_LoadUserInfo(FILENAME,
//  out userlD, out firstName, out lastName);
System.Windows.Forms.MessageBox.Show("Готово! " +userlD.ToString() + ",
" + lastName + ", " + firstName);
```

Лабораторная работа № 6. Разработка пользовательского интерфейса.
Модель однонаправленного чтения-записи XML-данных

В отличие от подхода XML DOM, обеспечивающего произвольный доступ к XML-данным и интенсивно использующего информацию о состоянии, объекты *XMLReader* и XMLWriter обеспечивают лишь возможности однонаправленного доступа. Они поддерживают минимальный объем информации о состоянии, которого достаточно для чтения и записи XML-данных, и не пытаются создавать в памяти дерево XML-данных или работать с ним. В этом случае говорят о моделях однонаправленного доступа, поскольку они поддерживают программный курсор, указывающий на текущую позицию в XML-файле, и позволяют работать только с находящимися в этом месте данными: курсор может перемещаться только в прямом направлении, но не в обратном. Объект *XMLReader* предлагает много возможностей, но в приложениях используется в основном для прохождения узлов XML-документа. При чтении XML-документа *XMLReader* каждый раз считывает только один узел и связанные с ним атрибуты; это напоминает чтение обычного файла по одной строке за один раз. Когда разработчик заканчивает просмотр узла и его атрибутов, он отдает объекту *XMLReader* команду перейти к следующему элементу, в результате чего *XMLReader* сбрасывает содержащуюся в нем информацию, которая относится к содержимому текущего узла. Однонаправленность доступа является необходимой платой за достижение наилучшей производительности и снижение накладных расходов

Следует отметить, что XML DOM строится поверх классов *XMLReader* и XMLWriter. XML DOM использует *XMLReader* для синтаксического анализа XML-документа и создает в памяти дерево на основе считанных данных. При записи XML-документа DOM итеративно проходит узлы находящегося в памяти дерева и выводит их через XMLWriter в

поток или файл. Как следствие, все, что можно сделать с использованием XML DOM, можно сделать, используя *XMLReader* и XMLWriter. XML DOM делает все возможное для того, чтобы с наибольшей эффективностью выполнять функции универсального XML-анализатора с произвольным доступом, максимально использующего информацию о состоянии.

Преимущества использования объектов *XMLReader* и XMLWriter вместо XML DOM коренятся в оптимизации, которая становится возможной либо благодаря тому, что вашему приложению не требуется универсальный синтаксический анализатор, либо благодаря тому, что работу можно выполнить, храня в памяти меньший объем информации о состояния, поскольку вам может не требоваться запись в файл всего XML-дерева, в которое были считаны XML-данные. Если нет нужды в использовании всех богатых функциональных возможностей XML DOM, то *XMLReader* и XMLWriter позволят добиться лучшей производительности, работая на более низком уровне абстракции.

Класс XmlTextReader обеспечивает быстрое однонаправленное чтение потока XML-данных. Данные могут быть получены из файла, объекта потока Stream или объекта TextReader. XmlTextReader обычно применяется если нужно считать XML документ и получить из него данные. Так как XmlTextReader не загружает весь документ в память, он является наилучшим выбором при обработке больших XML файлов - логов, дампов БД и пр.

Прочитаем данные с помощью класса XmlTextReader из XML-файла, ранее. В данном случае в файле будет содержаться информация о товарах предприятия и заказах.

```
OpenFileDialog dlg = new OpenFileDialog();
dlg.Filter = "Файлы XML (*.xml)|*.xml"; if (dlg.ShowDialog() ! = DialogResult.OK)
    return;
XmlTextReader reader = null;
orders.Clear();
try
{
    reader = new XmlTextReader(dlg.FileName);
    reader.WhitespaceHandling = WhitespaceHandling.None; // пропускаем пустые узлы
    while (reader.Read())
        if (reader.NodeType == XmlNodeType.Element)
            if (reader.Name == "Заказ")
            {
                Order order = new Order(reader.GetAttribute("Адрес"),
                DateTime.Parse(reader.GetAttribute("Дата")));
                // получаем товары в заказе
                while (reader.Read() && reader.Name == "Товар")
                    order.AddGood(reader.GetAttribute("Название"),
                    float.Parse(reader.GetAttribute("Цена")));
                orders.Add(order);
            }
            ShowOrders();
}
catch (Exception ex)
{
    MessageBox.Show("Ошибка: " + ex.Message);
}
finally
{
    if (reader != null)
        reader.Close();
}
```

XML-данные, которые читает XmlTextReader, берутся из файла, выбранного пользователем в диалоге. Для подавления пустых строк мы устанавливаем значение None для свойства WhitespaceHandling. Метод Read()производит чтение из потока следующего узла XML-документа. Он возвращает true если удалось считать узел. Обязательно нужно вызвать метод Read перед первым обращением к данным, т.к. в момент инициализации XmlTextReader не содержит никаких данных.

Разработчики уделяют огромное внимание поиску реализаций алгоритма, которые позволяют добиться максимального быстродействия. А вот о том, стоит ли вообще выполнять данную работу на мобильном устройстве, они чаще всего не задумываются. Во многих случаях некоторая работа может быть выполнена еще до того, как данные поступят на устройство, или переложена на сервер и выполнена в ответ на запрос. Располагая большими объемами доступной памяти, мощными процессорами и накопителями, серверы могут с успехом выполнять значительную часть необходимой предварительной работы, а также обработку по требованию, что можно использовать с выгодой для нужд мобильных приложений.

Лучше всего обрабатывать XML-данные на сервере еще до того, как они поступят на устройство. Если приложение ориентировано на использование данных, прошедших предварительную сортировку, фильтрацию и преобразования, то выполнение этой работы на сервере, прежде чем данные попадут на устройство, может принести вам реальные дивиденды в плане производительности. Эта задача заслуживает того, чтобы вы направили на нее часть своей творческой энергии.

Лабораторная работа №7. Разработка модели данных.

Целью лабораторной работы является изучить и использовать *инструментарий*, который применяется при отладке и развертывании мобильных приложений на *Windows Mobile*

Задание на лабораторную работу

1. Изучить и поэкспериментировать с удаленным инструментарием в частности

a. Remote *File Viewer*
b. Remote Heap Walker
c. Remote Process Viewer
d. Remote Registry Editor
e. Remote Spy
f.Remote Zoom In

2. Установить Power Toys 3.5, подключится к устройству и эмуляторам Visual Studio 2008. Поэкспериментировать с данным инструментарием, в частности
   a.Remote Performance Monitor
   b.GC Heap Viewer, NETCF CLR Profiler
   c.App *Configuration Tool* (NetCFcfg.exe)
   d.NETCF ServiceModel Metadata Tool
   e.Remote Logging *Configuration Tool*
   f.NETCF Network Log Viewer
3. Создание установочного cab файла.
4. Выводы

В данной лабораторной работе рассказывается о различном инструментарии который применяется при разработке, отладке и развертывании *Windows Mobile* приложений, а так же рассматриваются файлы журналов, которые может создавать исполнительная среда Microsoft *.NET Compact* Framework, об их использовании для диагностики конкретных проблем, об имеющемся для этого инструментарии, включения и отключения введения журналов. Журналы обеспечивают *протоколирование* активности установленного приложения, чтобы персонал службы поддержки имел некое средство диагностики.

Подключение к устройству.

Перед тем как обсуждать приемы отладки, оптимальные подходы к обработке исключений и рекомендации по поиску и устранению типичных проблем, нужно понять, как подключить Microsoft Visual Studio к целевому устройству, которым может быть как реальное устройство, так и эмулятор.

Удаленный инструментарий.

Visual Studio 2008 предоставляет разработчикам управляемых и неуправляемых кодов единые *средства разработки*. Ранее разработчикам неуправляемых кодов приходилось использовать встроенную интегрированную среду разработки Microsoft *Visual* C++. Одним из преимуществ такой унификации является то, что удаленный *инструментарий*, который ранее входил в *Visual* C++, но отсутствовал в *Visual Studio .NET* 2003, теперь включен в *Visual Studio* начиная с версии 2005. Теперь данные *инструментарий* имеется и в *Visual* Studio 2008, и доступен обеим группам разработчиков (рис.1). Так же следует отметить что данный *инструментарий* устанавливается автоматический с *Visual Studio* 2008.

Рис.1. Инструментарии удаленного доступа Visual Studio 2008

Программа Remote Zoom In очень удобна для захвата растровых экранных снимков с устройства. Часто используемой программой является *Remote Registry Editor*. Она позволяет разработчику с удаленного компьютера получить *доступ* к реестру и изменить его параметры. *Программа Remote File Viewer* является альтернативой Microsoft ActiveSync (или *Windows Mobile Device Center*) для удаленного доступа к файловой системе целевого устройства, включая импорт и экспорт. *Программа Remote* Spy аналогична настольной версии - с ее помощью можно просматривать активные окна и сообщения, которые посылаются описателям окон; эта *программа* не на каждый день, но если она действительно нужна, то может быть очень удобной. *Программа Remote* Process *Viewer* полезна для просмотра списка процессов, выполняющихся в данный момент на устройстве, принадлежащих им программных потоков, а также модулей, которые загрузил каждый из них. Её также можно использовать для остановки процессов. Последней программой является *Remote Heap* Walker. С точки зрения разработки управляемых кодов польза от нее невелика, так как разработчикам обычно не приходится иметь дело с идентификаторами и флагами для кучи управляемых процессов.

Удаленный *инструментарий* облегчает разработку программ на одном компьютере (настольном) и их выполнение на другом (целевом устройстве).

Задание первое.

В рамках данной лабораторной поэкспериментируйте с подключением к устройству с установленной ОС Windows Mobile 6.0 и к эмулятору Windows Mobile 6.

Power Toys 3.5

Следует скачать с сайта microsoft, на момент написания лабораторной работы, установочный файл для этого набора утилит можно было скачать по следующей ссылке: http://www.microsoft.com/downloads/details.aspx?FamilyID=c8174c14-a27d-4148-bf01-86c2e0953eab&displaylang=en

Набор утилит Power Toys 3.5 включено: Remote Performance Monitor and GC Heap Viewer, NETCF CLR Profiler, App *Configuration Tool* (NetCFcfg.exe), NETCF ServiceModel Metadata Tool, Remote Logging *Configuration Tool* и NETCF Network Log Viewer.

Remote Performance Monitor and GC Heap Viewer - предоставляет показатели производительности в реальном времени (деятельность сборщика мусора и типы загрузки данных), приложений запущенных на .NET Compact Framework. The GC Heap

Viewer feature позволяет вам захватывать состояние управляемой кучи в любой момент исполнения вашего приложения для просмотра текущих ссылок, и позволяет вам сравнить множество состояний, таким образом выявить момент утечки памяти.

NETCF CLR Profiler - это инструмент подробно визуализирует сведения о распределении ресурсов, показывает стеки вызовов и помогает диагностировать трудности в управлении памятью.

Утилита App *Configuration Tool* (NetCFcfg.exe) описывается авторами, как инструмент, который позволяет указать приложению, в какой версии среды .NET CF оно будет работать. Еще эта утилита выводит список установленных версий .NET CF и сведения об используемых библиотеках DLL. В справке к данной утилите сказано что, она устанавливается на мобильное устройство при первой же синхронизации в папку \Windows. Однако по факту это не всегда так, возможно вам в ручную придется её скопировать на ваше устройство, а её расположение следующее:

C:\Program Files\Microsoft.NET\SDK\CompactFramework\v3.5\WindowsCE\wce500\armv4i

В вашем случае вместо: диска C напишите диск с папкой Windows, wce500 - тип вашей платформы (данная папка соответствует WM 6), armv4i - тип процессора вашего процессора.

Утилита Remote Logging *Configuration Tool* позволяет настраивать параметры ведения журнала событий на целевом устройстве .NET CF, в том числе журналы загрузчика, взаимодействий, сети, ошибок и завершения программ. В рамках данной утилиты следует знать то, что необходимо указать, то в какой каталог должны размещаться файлы журналов.

Утилита NETCF Network Log Viewer предназначена просмотра содержимого из журнала работы сетевых соединений в среде .NET CF.

Так же необходимо указать, что Power Toys 3.5, имеет только английскую версию, и заработала корректно с эмуляторами Visual Studio 2008 только с английской Visual Studio 2008 SP1. Возможно, это ошибочное мнение, однако на 5 машинах устанавливалось все с нуля. И на четырех из пяти, устанавливались русские программные продукты Microsoft, на которых Power Toys 3.5 не обнаруживала эмуляторов Visual Studio 2008, а пятая английская заработала сразу.

Данные инструменты легко применимы, те трудности, которые они могут вызваны, уже описаны. Так что далее опишем журналы, которые возможно включить в Remote Logging *Configuration Tool.*

Журнал загрузчика

Если немного упростить реальность, то в каждом случае, когда управляемое приложение создает объект, использует *значимый тип*, вызывает статический метод или делает что-то подобное, исполнительная среда должна найти сборку, которая содержит данный тип, и загрузить его. Каждый раз, когда это не получается, запускается исключение. В вопросе о том, следует ли его пропустить (поглотить или замаскировать другим исключением), может помочь журнал. Если исполнительная среда не нашла член *типа (исключение* Missing-MethodException или MissingFieldException) или сам *тип (исключение* TypeLoadException), вы сможете увидеть хронологию загрузки сборок вместе с той, которая привела к сбою, что в конечном счете и породило одно из упомянутых исключений времени выполнения. Информация о загрузке сборок включает информацию о маркерах открытых ключей, версиях и путях, по которым среда искала сборку. Другими словами, вы можете получить богатую информацию, которой больше нигде не найти. Если вы знакомы с журналом слияний для настольного компьютера - это почти то же самое.

В качестве упражнения посмотрите сами, как выглядит журнал загрузчика. Просто запустите любое управляемое приложение на устройстве и скопируйте файл журнала для его изучения, на настольный компьютер.

Задание второе

1. В проекте приложения для интеллектуального устройства сделайте ссылку на проект библиотеки классов интеллектуальных устройств. В обработчике события Click для кнопки создайте класс из библиотеки классов.
2. Разверните и отладьте проект на устройстве с помощью Visuai Studio. Получите файл журнала.
3. На устройстве перейдите в папку приложения и удалите DLL-файл. Запустите непосредственно EXE-файл, нажмите кнопку и понаблюдайте за аварийным завершением приложения. Получите файл журнала и обратите внимание па различия с полученным рапсе журналом.

Журнал взаимодействия с платформой

Несмотря на то что в .NET Compact Framework версии 3.5 заполнены многие пробелы версии 2.0 и 1.0, разработчикам время от времени необходимо вызывать неуправляемые Windows-методы из DLL-файлов при помощи платформенных сервисов вызова (PInvoke-сервисов). Когда возникают проблемы с PInvokc-сервисом (или при взаимодействии с COM), может возникать множество исключений (например, DllNotFound-Exception, EntryPointNotFoundException), в других случаях ошибки маршалинга могут просто привести к неверным результатам. Журнал взаимодействия с платформой помогает выявлять ошибки маршалипга, так как дает список всех методов взаимодействия в управляемом варианте и его неуправляемом эквиваленте. Иногда неуправляемый эквивалент может помочь разработчикам распознать неверное объявление.

Задание третье

Запустите па устройстве любое управляемое приложение, которое делает PInvoke-вызовы, и скопируйте файл журнала на настольный компьютер для изучения. Приложение, написанное в лабораторной работе "Применение платформы MS Synchronization Services при создании приложений без постоянного соединения" использовало PInvokc-вызовы.

Посмотрите содержимое журнала взаимодействия сами.

Сетевой журнал

Сетевые журналы собирают богатую информацию о сетевой активности. В отличие от других журналов они содержат данные не только в формате ASCII, поэтому для декодирования имеющихся в них двоичных данных нужна программа - анализатор. Такая программа для .NET Compact Framework 3.5 имеется в пакете программ .NET Compact Framework Power Toys 3.5.

Задание четвертое

Скопируйте с устройства на персональный компьютер файл, который создает платформа в результате работы программы "Применение платформы MS Synchronization Services". Просмотрите его с помощью файла из лабораторной работы применение платформы MS Synchronization Services . Вы можете открыть с ее помощью файл сетевого журнала и изучить его содержимое, то есть пакеты, которые посылаются и принимаются.

Лабораторная работа №8. Развертывание мобильного приложения.

Подключение к отладчику

В некоторых случаях требуется отладить *приложение*, которое выполняется на мобильном устройстве, но не было запущено из *Visual Studio*. В этом случае нужно подключиться к выполняемому па устройстве процессу из *Visual Studio*. Это возможно, по только при явном включении соответствующего режима, например, через реестр. При подключенном устройстве используйте удаленный редактор реестра для создания под ключом HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETCompactFraiuework\ ключа с названием Managed *Debugger*. Под этим ключом создайте *значение* AttachEnabled типа DWORD и присвойте ему 1.

Затем в *меню* Debug в *Visual Studio* выберите *пункт* Attach To Process и в открывшемся диалоговом окне измените Transport на Smart Device, а в раскрывающемся списке Qualifier выберите ваше устройство. Далее в списке выберите ваш управляемый процесс и затем щелкните на кнопке Attach.

Как и *протоколирование*, этот режим отрицательно влияет на *производительность* всех управляемых приложений на устройстве, поэтому подключайтесь к отладчику только при отладке приложения.

Журнал ошибок

Журнал ошибок можно создавать, только начиная с версии .NET Compact Framework 2.0 SP1 и в более поздних. Он был добавлен для помощи в отладке безмониторных устройств, но его, естественно, можно также использовать и для отладки устройств с дисплеями. Журнал ошибок - это просто текстовая версия этого диалогового окна "error list".

Утилиты для создания программы установки в Visual Studio 2008

В *Visual Studio* 2008 вводится новый *шаблон* проекта специально для создания программы установки в виде CAB-файла непосредственно из интегрированной среды разработки. Раньше можно было создать CAB-*файл* программы установки только однажды, а затем приходилось вручную изменять INF-*файл* для повторной генерации CAB-файла при любых изменениях параметров. При новом подходе, этот процесс больше напоминает создание программы установки для настольного проекта.

Проект создания программы установки для мобильного устройства

Чтобы создать новую программу установки для мобильно устройства, в *меню* File выберите *пункт* New Project. Перейдите но цепочке Other Project Types, затем Setup And Deployment и выберите проект Smart Device CAB Project. Выбрав проект в дереве решений, можно настроить множество параметров проекта. Например, свойства Manufacturer и ProductName используются вместе в качестве имени компонента в списке Remove Programs. Поэтому рекомендуется, чтобы общая *длина* составной строки имени компонента не превышала примерно 36 символов, так как на большинстве устройств с портретной ориентацией строка будет обрезана. В том редком случае, когда вы хотите запретить пользователю удалять ваше *приложение*, можете исключить *приложение* из списка Remove Programs путем установки свойства NoUninstall в *значение* True.

Добавление файлов и настройка целевого устройства

Файлы устанавливаются путем вставки их в редакторе файловой системы нашего проекта программы установки. Вы можете использовать множество констант стандартных папок, которые развертывают свое содержимое в нужной локализованной папке на устройстве. Вы можете также создавать новые каталоги и подкаталоги.

Рассмотрим полученные данные на практическом примере, создания установочного cab-файла, на примере имеющегося проекта MyFirstSync. Создадим проект Smart Device CAB Project находящийся в Other Project Types, задав ему имя MySyncProgram. И перед вами появится окно (рис.2).

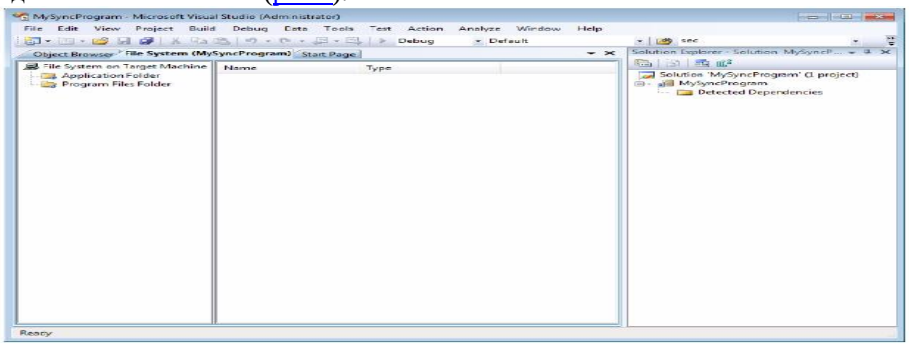


Рис.2. Окно проекта для создания CAB файла

Теперь в папке Program Files Folder создадим папку MySyncProgram и в нее следует добавить файлы, находящиеся в папке Bin\Release проекта, для которого создаем установочный файл. В папке Bin\Release, находится необходимый исполняемый файл и файл баз данных, их следует добавить в вашу папку, а dll файлы можно не добавлять. Так же общая рекомендация, если вы затрудняитесь в выборе добавляемых файлов, то выберите те файлы которые Visual Studio 2008, размещает в процессе Debug в папку приложения на эмуляторе.

После этого увидим MySyncProgram в *Solution Explorer* проекта развертывания. В разделе *Detected Dependencies* (Обнаруженные зависимости) увидим, что *Visual Studio* опросил MyFirstSync.exe на предмет сборок, которых он зависит - в данном случае это два dll файла (*Microsoft.Synchronization.Data.DLL* и *Microsoft.Synchronization.Data.SqlServerCe.DLL*). Ваше приложение будет работать, только если на устройстве установлен .net Compact Framework 3.5 и Compact SQL 3.5. Таким образом их не следует включать.

Далее следует нажать правой кнопкой мыши в области окна файловой системы и выбрать подменю Start Menu Folder меню Add Special Folder (рис.3).

Добавление ярлыков

Ярлыки - это просто файлы, и в качестве таковых они также добавляются при помощи редактора файловой системы. Для создания ярлыка файла и проекте щелкните правой кнопкой мыши на файле и выберите команду Create Shortcut. Затем

можете переименовать ярлык, чтобы присвоить ему удобочитаемое имя, и поместить его в нужную папку, например, в папку Start Menu Folder, если вы хотите, чтобы он присутствовал в подменю Programsменю Start.

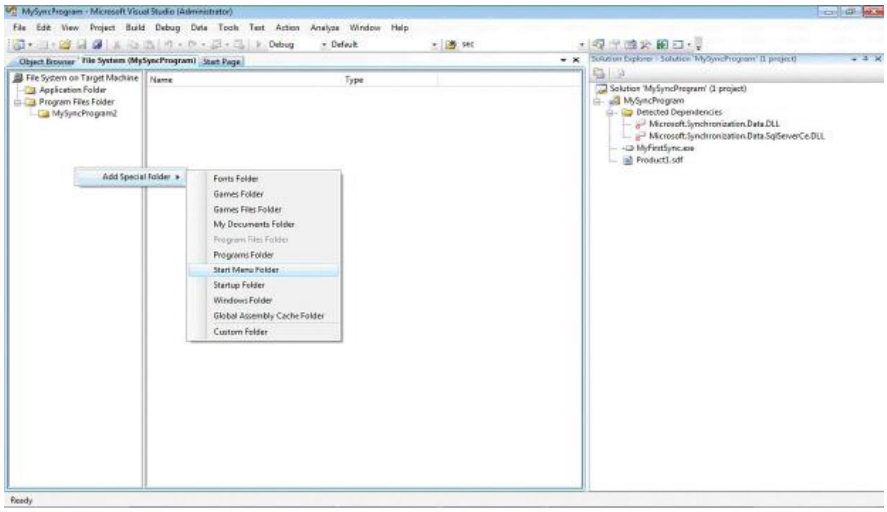


Рис.3. Добавление специальных папок

Запись параметров реестра

Так же как и в проектах программ установки настольных версий *Windows*, в проекте развертывания на устройстве есть встроенная *поддержка* записи параметров реестра. Выберите редактор реестра, чтобы увидеть *дерево* корневых ключей. Под ними можете создавать собственные ключи. В том случае, если на устройстве их еще нет, они будут созданы. Часто эти ключи создаются не инсталлятором, а самой программой, в процессе работы пользователя с ней. Однако наша *программа* не столько сложна и не требует записей в реестр.

Теперь осуществим сборку (F6), и необходимый вам cab *файл* будет расположен в папке DEBUG вашего CAB проекта.

Выводы

Вы познакомились с эффективными средствами диагностики и средствами управления состояния *Windows Mobile* на устройстве или эмуляторе *Visual Studio* 2008 и узнали как избежать типичных трудностях начинающих программистов в освоении данного инструментария.

Комплект заданий для проведения тренинга

Тренинг 1.

Форма, кнопка, метка и диалоговое окно. Событие MouseHover.

После установки системы программирования Visual Studio 2010, включающей в себя Visual Basic 2010, целесообразно создать ярлык на рабочем столе для запуска программы. После запуска программы мы видим начальный пользовательский интерфейс, показанный на рис.1.1.

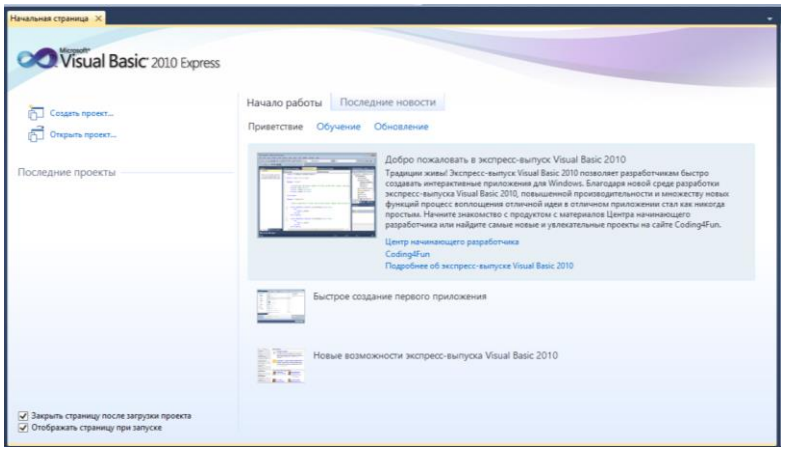


Рис.1.1. Стартовая страница системы Visual Basic 2010 Express.

Для начала программирования, необходимо в пункте меню Файл выполнить команду Создать проект. В открывшемся окне, в левой колонке, находится список встроенных шаблонов (Installed Templates). Среди них – шаблоны языков программирования, встроенных в Visual Studio. Нам нужен Visual Basic. В средней колонке выберем шаблон Приложения Windows Forms и нажмем кнопку ОК. В открывшемся окне рис.1.2., изображена экранная форма – Form1, в которой программисты располагают различные компоненты графического интерфейса пользователя или элементы управления с панели Свойства.

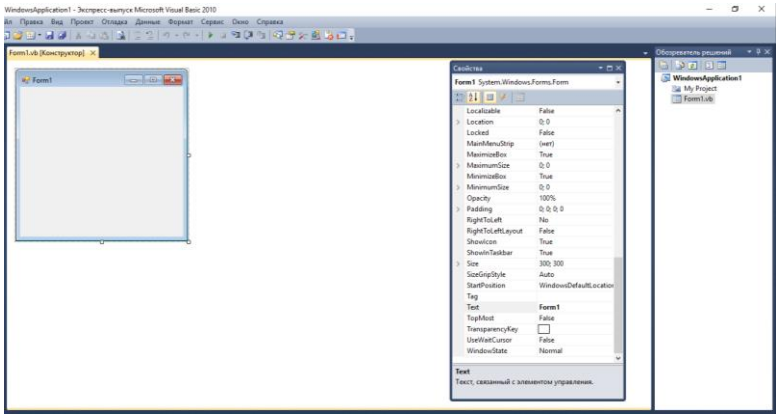


Рис.1.2. Окно для проектирования пользовательского интерфейса.

Задание 1. Написать программу, которая будет отображать такую экранную форму, в которой будет написано «Microsoft Visual Basic 2010», также в форме будет расположена командная кнопка с надписью «Нажми меня». При нажатии кнопки должно появляться диалоговое окно с сообщением «Всем привет».

Ход действий.

В программе все, что может быть названным именем существительным, называют объектом. В нашей программе мы имеем 4 объекта: форму Form, надпись на форме Label, кнопку Button и диалоговое окно MessageBox с текстом «Всем привет».

Добавьте метку и кнопку на форму, как показано на рис.1.3.

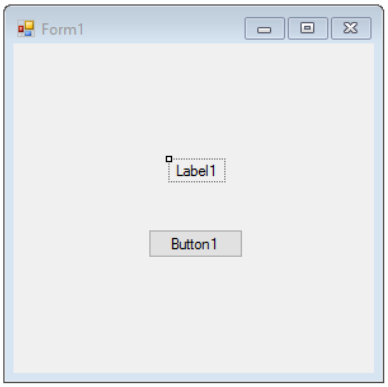


Рис.1.3. Форма первого проекта.

В нашем задании мы будем пользоваться готовыми визуальными объектами, которые можно перетаскивать мышью из панели Свойства. Каждый объект имеет свойства рис.1.4. Свойства можно увидеть, если щелкнуть правой кнопкой мыши в пределах формы и выбрать в контекстном меню команду Свойства.

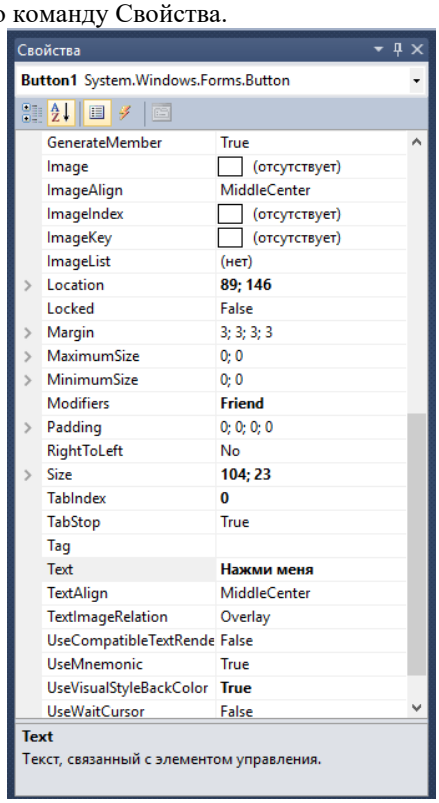


Рис. 1.4. Свойства кнопки Button1

Для объекта Label1 выберем свойство Text и напишем напротив этого поля «Microsoft Visual Basic 2010». Для объекта Button1 – «Нажми меня».

Все объекты обрабатываются событиями. Событием является щелчок на кнопке, щелчок в пределах формы, загрузка формы в оперативную память и т.д. В нашем задании событием является щелчок по командной кнопке. Результатом щелчка должно появиться диалоговое окно с надписью «Всем привет!».

Перейдем на вкладку для написания кода: щелчок правой кнопкой мыши в пределах формы, затем выбрать команду Перейти к коду. В раскрывшемся списке перечислены объекты, которые присутствуют в данном проекте. Выберем Button1.

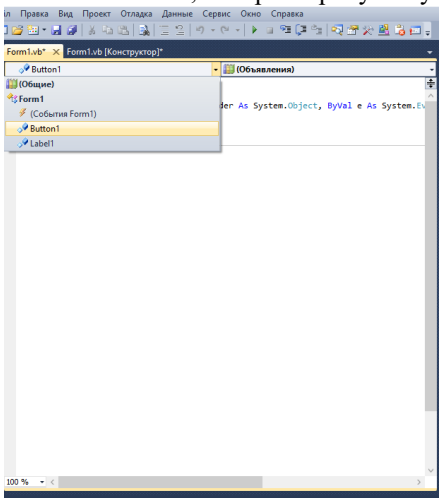


Рис. 1.5. Список объектов.

В раскрывшемся списке выберем событие Click.

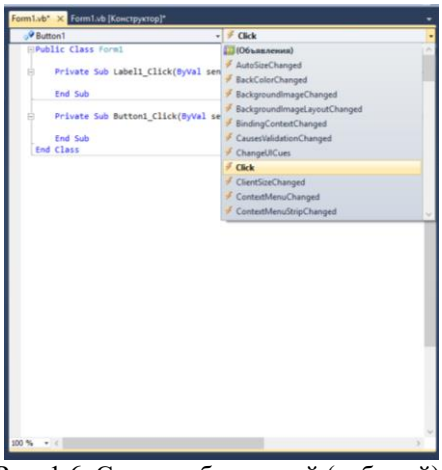


Рис. 1.6. Список объявлений (событий).

При этом управляющая среда Visual Basic 2010 генерирует две строчки программного кода (рис.1.7).

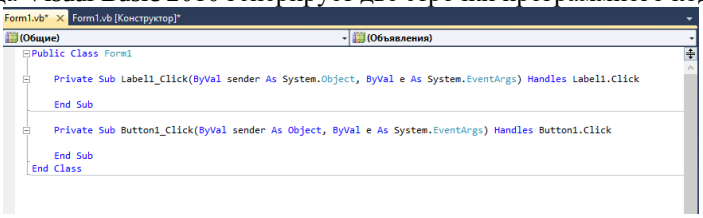


Рис.1.7. Вкладка программного кода.

Таким образом, система написала начало процедуры Sub обработки события Button1_Click и конец процедуры End Sub. Эти две строчки называют пустым обработчиком события. Заполним этот обработчик. Для этого между этими строчками напишем:

MessageBox.Show (“Всем привет!”)

Объекты кроме свойств имеют также и методы, т.е. программы, которые обрабатывают объекты. Так для объекта MessageBox вызывается метод Show.

В этих трех строчках мы написали *процедуру обработки события* нажатия кнопки Button1. Для проверки работоспособности программы нажмем клавишу F5 (рис.1.8.).

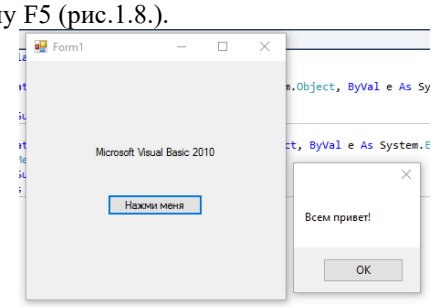


Рис.1.8. Работающая программа.

Задание 2. Событие MouseHover.

Немного усложним предыдущую задачу. Добавим еще одну обработку события MouseHover мыши для объекта Label1. Событие MouseHover наступает тогда, когда пользователь указателем мыши «зависает» над каким-либо объектом, причем именно «зависает», а не просто перемещает мышь над объектом. Есто еще событие MouseEnter (Войти), когда указатель мыши *входит в пределы* области элемента управления (в данном случае метки Label1).

Переключимся на вкладку программного кода Form1.vb. У нас есть две вкладки: Form1.vb и Form1.vb[Конструктор], т.е. вкладка программного кода и вкладка визуального проекта программы. Переключение между ними осуществляется мышью или комбинацией клавиш <Ctrl>+<Tab>.

Запрограммируем событие MouseHover объекта Label1. В окне редактора в левом верхнем раскрывающемся списке выбираем объект label1, а в правом – событие MouseHover. При этом среда VB2010 генерирует две строки программного кода (пустой обработчик):

Private Sub Label1_MouseHover {*Параметры процедуры…*}

End Sub

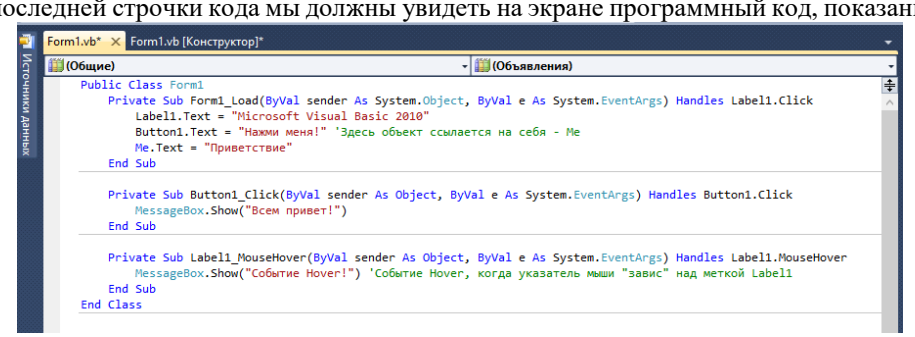
Между этими двумя строчками вставляем вызов диалогового окна:

MessageBox.Show (“Событие Hover!”)

Произведем проверку программы: нажимаем клавишу F5, «зависаем» указателем мыши над Label1, щелкаем по кнопке Button1.

Изменим в строке заголовка надпись Form1 на слово «Приветствие». Для этого ниже присваиваем эту строку свойству Text формы. Поскольку мы изменяем свойство объекта Form1 внутри подпрограммы обработки события, связанного с формой, следует к форме обращаться через ссылку Me: Me.Text = “Приветствие” или MyBase.Text = “Приветствие”.

После написания последней строчки кода мы должны увидеть на экране программный код, показанный в листинге 1.1.



Листинг 1.1. Программирование событий.

Комментарии, поясняющие работу программы, в окне редактора кода будут выделены зеленым цветом. В VB комментарии пишут после одиночной кавычки (‘) или после ключевого слова REM.

Сохраним и закроем проект под именем Hover (Файл – Закрыть проект). Теперь программный код этой программы можно посмотреть, открыв решение Hover.sln, в папке Hover.

Комплект заданий для самостоятельной работы обучающихся

Вопросы для подготовки к занятиям

1. Какое событие генерируется на этапе выгрузки страницы?
2. Какие языки разметки поддерживаются управляющими элементами ASP.NET Mobile Controls?
3. Из каких компонентов состоит значение атрибута?
4. Какой режим работы устройства Bluetooth предусматривает регулярные обмены с целью синхронизации клиентов?
5. Что является недостатком беспроводных сетей?
6. Что такое том в контексте использования EDB или CEDB?
7. Какие данные являются пользовательскими данными?
8. Какой компонент сети GPRS является блоком управления пакетами, дающим возможность станциям GSM пересылать и получать пакеты при GPRS коммуникациях?
9. Что описывает модель доступа к данным?
10. Что является достоинством локальных баз данных?
11. Какие элементы управления являются списочными?
12. В каких случаях можно создать журнал ошибок?
13. Чем определяется выбор наиболее подходящей модели хранения данных в памяти?
14. В каком случае исполнительная среда должна найти сборку, которая содержит требуемый тип, и загрузить его?
15. Какой вариант установки приложения на мобильное устройство является аналогом установки программного обеспечения на настольных компьютерах с компакт-дисков, когда инсталляционная программа автоматически запускается после вставки компакт-диска или диска DVD в соответствующий привод?
16. Сколько активных узлов может поддерживать главный узел пикосети?
17. Какие недостатки имеют устройства с сенсорным экраном?

Комплект тестовых заданий

Тест 1.

1. Какие утверждения о беспроводных сетях являются верными?
   - беспроводные сети не чувствительности к радиопомехам
   +позволяют избежать прокладки кабеля внутри здания

+сеть можно очень быстро развернуть

2. Между какими устройствами может быть установлена связь с помощью технологии Bluetooth?
   +принтерами
   +мобильными телефонами
   +стационарными PC
3. В каком режиме устройство-клиент имеет наивысшую скважность рабочего цикла (наименьшую экономию энергии)?
   -Hold
   -Active
   -Park
   +Sniff
4. Сколько активных узлов может поддерживать главный узел пикосети?
   +до 7
   -до 255
   -только 1
5. Какие действия выполняются на уровне радиосвязи протокола L2CAP?
   -преобразование потока бит в кадры
   +передача данных от главного узла к подчиненному бит за битом
   -определение базовых форматов
6. Что такое FTP (File Transfer Profile)?
   -протокол определения предлагаемых сервисов
   -протокол связи мобильной ЭВМ со стационарной LAN
   -протокол связи мобильного факса с мобильным телефоном
   +протокол пересылки файлов
7. Какой протокол предназначен для определения услуг, оказываемых удаленным устройствам?
   +SDP (Service Description Protocol)
   -PAN (Personal Area Networking)
   -SPP (Serial Port Profile)
8. Какие утверждения справедливы для технологии GPRS?
   +перед передачей данные собираются в пакеты
   +пакеты передаются по незанятым голосовым каналам
   -скорость передачи данных постоянно высокая
9. Какой компонент сети GPRS отвечает за взаимодействие с Интернет и другими общественными сетями, передающими данные и голос?
   -SGSN
   +GGSN
   -GTP
   -PCU
10. Какие функции выполняет Microsoft Sync Framework?
    +обеспечивает автономный доступ к данным приложений
    +обеспечивает автономный доступ к данным служб и устройств
    +обеспечивает коллективную работу
11. Какие утверждения являются неверными?
    -в SQL Server Compact 3.5 строка подключения является путем к SDF-файлу базы данных
    +SQL Server Compact 3.5 запускается как служба
    +SQL Server Compact 3.5 поддерживает только одно подключение
12. Какая из перечисленных сред поддерживает .NET Compact Framework 3.5?
    -Visual Studio .NET 2003
    -Visual Studio 2005
    +Visual Studio 2008

Тест 2.

1. Какие средства входят в состав библиотек пользовательского интерфейса?
   -средства, позволяющие учитывать в приложениях региональную и местную специфику
   -средства файлового ввода-вывода
   +средства создания двумерных рисунков
2. Какое событие генерируется на этапе выгрузки страницы?
   -Load
   -PreRender
   +Unload
3. Какой класс используется для однонаправленной записи XML-документа?
   -XmlDocument
   -XmlReader
   +XmlWriter
4. Какой класс используется для однонаправленного чтения XML-документа?
   -XmlWriter
   +XmlReader
   -XmlDocument
5. Что происходит на этапе загрузки состояния представления мобильной страницы?
   -определяется адаптер устройства, который будет использоваться при формировании вывода страницы

+восстанавливается сохраненное состояние страницы
-страница загружает входящие данные формы, кэшированные в объекте Request, и соответствующим образом обновляет свои свойства

6. Какие недостатки имеют устройства с сенсорным экраном?
-нетребовательны условиям хранения
+требовательны условиям хранения
-имеют менее широкие возможности в плане элементов управления в силу ограниченности механизма взаимодействия
-имеют более широкие возможности в плане использование элементов интерфейса
7. Какие утверждения являются неверными?
+процедура упаковки одинакова для всех технологий (например, для /J2SE, .NET Compact Framework)
-для разных типов устройств, предусмотрены различные процедуры инсталляции программного обеспечения
-детали упаковки и развертывания приложений определяются спецификой используемых типов устройств и программных технологий
8. Какие утверждения являются верными?
-в версии Visual Studio 2008 Professional отсутствуют функции разработки для мобильных устройств
+Visual Studio 2005 позволяет проектировать приложения для мобильных устройств
+при создании нового проекта SmartDevice в Visual Studio 2008 в качестве целевой платформы можно выбрать .NET Compact Framework 2.0
9. Какие элементы управления являются списочными?
+ComboBox
+ListBox
-InputPanel
-TextBox
10. Какой элемент представляет собой выпадающий список?
-InputPanel
-ListBox
+ComboBox
-TextBox
11. На каком этапе жизненного цикла мобильной страницы страница генерирует вывод, который будет направлен клиенту?
-инициализация страницы
+рендеринг страницы
-сохранение состояния представления
12. Какое событие генерируется на этапе загрузки пользовательского кода?
-Unload
-PreRender
+Load

Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций в процессе освоения образовательной программы

Критерии оценки к экзамену
Оценка «отлично» (86-100 баллов) ставится обучающемуся, обнаружившему систематические и глубокие знания учебно -программного материала, умения свободно выполнять задания, предусмотренные программой в типовой

ситуации (с ограничением времени) и в нетиповой ситуации, знакомство с основной и дополнительной литературой, усвоение взаимосвязи основных понятий дисциплины в их значении приобретаемой специальности и проявившему творческие способности и самостоятельность в приобретении знаний. Студент исчерпывающим образом ответил на вопросы экзаменационного билета. Задача решена правильно, студент способен обосновать выбранный способ и пояснить ход решения задачи.

Оценка «хорошо» (71-85 баллов) ставится обучающемуся, обнаружившему полное знание учебно-программного материала, успешное выполнение заданий, предусмотренных программой в типовой ситуации (с ограничением времени), усвоение материалов основной литературы, рекомендованной в программе, способность к самостоятельному пополнению и обновлению знаний в ходе дальнейшей работы над литературой и в профессиональной деятельности. При ответе на вопросы экзаменационного билета студентом допущены несущественные ошибки. Задача решена правильно или ее решение содержало несущественную ошибку, исправленную при наводящем вопросе экзаменатора.

Оценка «удовлетворительно» (56-70 баллов) ставится обучающемуся, обнаружившему знание основного учебно-программного материала в объеме, достаточном для дальнейшей учебы и предстоящей работы по специальности, знакомство с основной литературой, рекомендованной программой, умение выполнять задания, предусмотренные программой. При ответе на экзаменационные вопросы и при выполнении экзаменационных заданий обучающийся допускает погрешности, но обладает необходимыми знаниями для устранения ошибок под руководством преподавателя. Решение задачи содержит ошибку, исправленную при наводящем вопросе экзаменатора.

Оценка «неудовлетворительно» (менее 56 баллов) ставится обучающемуся, обнаружившему пробелы в знаниях основного учебно-программного материала, допустившему принципиальные ошибки в выполнении предусмотренных программой заданий, слабые побуждения к самостоятельной работе над рекомендованной основной литературой.

Оценка «неудовлетворительно» ставится обучающимся, которые не могут продолжить обучение или приступить к профессиональной деятельности по окончании академии без дополнительных занятий по соответствующей дисциплине.

Критерии оценивания контрольной работы текущего контроля успеваемости обучающихся (рекомендуемое)

Комплект контрольных вопросов для проведения устных опросов

Критерии оценивания (устанавливаются разработчиком самостоятельно с учетом использования рейтинговой системы оценки успеваемости обучающихся)

Примерные критерии оценивания:

– правильность ответа по содержанию задания (учитывается количество и характер ошибок при ответе);

– полнота и глубина ответа (учитывается количество усвоенных фактов, понятий и т.п.);

– сознательность ответа (учитывается понимание излагаемого материала);

– логика изложения материала (учитывается умение строить целостный, последовательный рассказ, грамотно пользоваться специальной терминологией);

– использование дополнительного материала;

– рациональность использования времени, отведенного на задание (не одобряется затянутость выполнения задания, устного ответа во времени, с учетом индивидуальных особенностей обучающихся).

Шкала оценивания (устанавливается разработчиком самостоятельно с учетом использования рейтинговой системы оценки успеваемости обучающихся)

Примерная шкала оценивания:

Баллы для учета в рейтинге (оценка)	Степень удовлетворения критериям
86-100 баллов «отлично»	Обучающийся полно и аргументировано отвечает по содержанию вопроса (задания); обнаруживает понимание материала, может обосновать свои суждения, применить знания на практике, привести необходимые примеры не только по учебнику, но и самостоятельно составленные; излагает материал последовательно и правильно.
71-85 баллов «хорошо»	Обучающийся достаточно полно и аргументировано отвечает по содержанию вопроса (задания); обнаруживает понимание материала, может обосновать свои суждения, применить знания на практике, привести необходимые примеры не только по учебнику, но и самостоятельно составленные; излагает материал последовательно. Допускает 1-2 ошибки, исправленные с помощью наводящих вопросов.
56-70 баллов «удовлетво-рительно»	Обучающийся обнаруживает знание и понимание основных положений данного задания, но излагает материал неполно и допускает неточности в определении понятий или формулировке правил; не умеет достаточно глубоко и доказательно обосновать свои суждения и привести свои примеры; излагает материал непоследовательно и допускает
0-55 баллов «неудовлетворительно»	Обучающийся обнаруживает незнание ответа на соответствующее задание (вопрос), допускает ошибки в формулировке определений и правил, искажающие их смысл, беспорядочно и неуверенно излагает материал. Отмечаются такие недостатки в подготовке обучающегося, которые являются серьезным препятствием к успешному овладению последующим материалом.

Критерии оценивания контрольной работы для практических (лабораторных) работ

Критерии оценивания (устанавливаются разработчиком самостоятельно с учетом использования рейтинговой системы оценки успеваемости обучающихся)
Примерные критерии оценивания:
– правильность выполнения задания на практическую/лабораторную работу в соответствии с вариантом;
– степень усвоения теоретического материала по теме практической /лабораторной работы;
– способность продемонстрировать преподавателю навыки работы в инструментальной программной среде, а также применить их к решению типовых задач, отличных от варианта задания;
– качество подготовки отчета по практической / лабораторной работе;
– правильность и полнота ответов на вопросы преподавателя при защите работы
и др.
Шкала оценивания (устанавливается разработчиком самостоятельно с учетом использования рейтинговой системы оценки успеваемости обучающихся)

Примерная шкала оценивания практических занятий (лабораторных работ):

Баллы для учета в рейтинге (оценка)	Степень удовлетворения критериям
86-100 баллов «отлично»	Выполнены все задания практической (лабораторной) работы, обучающийся четко и без ошибок ответил на все контрольные вопросы.
71-85 баллов «хорошо»	Выполнены все задания практической (лабораторной) работы; обучающийся ответил на все контрольные вопросы с замечаниями.
56-70 баллов «удовлетво-рительно»	Выполнены все задания практической (лабораторной) работы с замечаниями; обучающийся ответил на все контрольные вопросы с замечаниями.
0-55 баллов «неудовлетворительно»	Обучающийся не выполнил или выполнил неправильно задания практической (лабораторной) работы; обучающийся ответил на контрольные вопросы с ошибками или не ответил на контрольные вопросы.

Критерии оценивания контрольной работы тестовых заданий

Материалы тестовых заданий
Материалы тестовых заданий следует сгруппировать по темам/разделам изучаемой дисциплины (модуля) в следующем виде:
Тема (темы) / Раздел дисциплины (модуля)
Тестовые задания по данной теме (темам)/Разделу с указанием правильных ответов.
Критерии оценивания (устанавливаются разработчиком самостоятельно с учетом использования рейтинговой системы оценки успеваемости обучающихся)
Примерные критерии оценивания:
- отношение правильно выполненных заданий к общему их количеству
Шкала оценивания (устанавливается разработчиком самостоятельно с учетом использования рейтинговой системы оценки успеваемости обучающихся)

Примерная шкала оценивания:

Баллы для учета в рейтинге (оценка)	Степень удовлетворения критериям
86-100 баллов «отлично»	Выполнено 86-100% заданий
71-85 баллов «хорошо»	Выполнено 71-85% заданий
56-70 баллов «удовлетво-рительно»	Выполнено 56-70% заданий
0-55 баллов «неудовлетворительно»	Выполнено 0-56% заданий

Критерии оценивания контрольной работы кейс-задач

Задание (я):
Критерии оценивания (устанавливаются разработчиком самостоятельно с учетом использования рейтинговой системы оценки успеваемости обучающихся)

Примерные критерии оценивания:
- соответствие решения сформулированным в кейсе вопросам (адекватность проблеме и рынку);
- оригинальность подхода (новаторство, креативность);
- применимость решения на практике;
- глубина проработки проблемы (обоснованность решения, наличие альтернативных вариантов, прогнозирование возможных проблем, комплексность решения).
Шкала оценивания (устанавливается разработчиком самостоятельно с учетом использования рейтинговой системы оценки успеваемости обучающихся)

Примерная шкала оценивания:

Баллы для учета в рейтинге (оценка)	Степень удовлетворения критериям
86-100 баллов «отлично»	Предложенное решение соответствует поставленной в кейс-задаче проблеме. Обучающийся применяет оригинальный подход к решению поставленной проблемы, демонстрирует высокий уровень теоретических знаний, анализ соответствующих источников. Формулировки кратки, ясны и точны. Ожидаемые результаты применения предложенного решения конкретны, измеримы и обоснованы.
71-85 баллов «хорошо»	Предложенное решение соответствует поставленной в кейс-задаче проблеме. Обучающийся применяет в основном традиционный подход с элементами новаторства, частично подкрепленный анализом соответствующих источников, демонстрирует хороший уровень теоретических знаний. Формулировки недостаточно кратки, ясны и точны. Ожидаемые результаты применения предложенного решения требуют исправления незначительных ошибок.
56-70 баллов «удовлетво-рительно»	Демонстрирует средний уровень знаний, умений, навыков в соответствии с критериями оценивания. Предложенное решение требует дополнительной конкретизации и обоснования, в целом соответствует поставленной в задаче проблеме. При решении поставленной проблемы обучающийся применяет традиционный подход, демонстрирует твердые знания по поставленной проблеме. Предложенное решение содержит ошибки, уверенно исправленные после наводящих вопросов.
0-55 баллов «неудовлетворительно»	Наличие грубых ошибок в решении ситуации, непонимание сущности рассматриваемой проблемы, неуверенность и неточность ответов после наводящих вопросов. Предложенное решение не обосновано и не применимо на практике

ИЗМЕНЕНИЯ И ДОПОЛНЕНИЯ

Ведомость изменений

№ п/п	Вид обновлений	Содержание изменений, вносимых в ОПОП	Обснование изменений
1			
2			
3			
4			
5			
6			