ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

СОВРЕМЕННЫЕ СИСТЕМЫ ПРОГРАММИРОВАНИЯ

Методические рекомендации к лабораторным работам для студентов направлений подготовки 09.03.04 «Программная инженерия» и 09.03.01 «Информатика и вычислительная техника» дневной формы обучения

Могилев 2018

УДК 681.3
ББК 32.973
С 68

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «9» октября 2018 г., протокол № 4

Составитель ст. преподаватель Н. В. Выговская

Рецензент Ю. С. Романович

Приведены методические рекомендации к лабораторным работам по дисциплине «Современные системы программирования» для студентов направлений подготовки 09.03.04 «Программная инженерия» и 09.03.01 «Информатика и вычислительная техника» дневной формы обучения.

Учебно-методическое издание

СОВРЕМЕННЫЕ СИСТЕМЫ ПРОГРАММИРОВАНИЯ

Ответственный за выпуск	А. И. Якимов
Технический редактор	А. А. Подошевко
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 31 экз. Заказ №

Издатель и полиграфическое исполнение:
Государственное учреждение высшего профессионального образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 24.01.2014.
Пр. Мира, 43, 212000, Могилев.

Содержание

Введение

Цель методических рекомендаций к выполнению лабораторных работ по дисциплине «Современные системы программирования» – дать студентам необходимый базовый и практический материал для овладения методами и технологиями построения современных систем обработки информации, выработать навыки использования передовых методов и технологий разработки программного обеспечения.

Целью преподавания дисциплины «Современные системы программирования» является формирование у студентов объективного взгляда на современную теорию и практику программирования, получение знаний о современных подходах к проектированию и реализации программных систем.

Рассматриваемый в данных методических рекомендациях материал содержит перечень лабораторных работ по дисциплине с минимальным количеством необходимых теоретических сведений и заданиями для самостоятельного выполнения.

После выполнения лабораторной работы студентом предоставляется преподавателю отчет, который содержит разделы:

- тема и цель работы;
- текст программы;
- результаты выполнения программы.

Полученные при изучении дисциплины знания и навыки могут быть востребованы при изучении учебных дисциплин «Технологии интернет-программирования», курсовом и дипломном проектировании и для реализации собственных проектов.

1 Лабораторная работа № 1. Разработка программ методом TDD

Цель работы: изучить методики гибкой (agile) разработки программного обеспечения и управления проектами на примере программирования на основе тестирования.

1.1 Краткие теоретические сведения

Разработка через тестирование (англ. *test-drivendevelopment*) – техника программирования, при которой модульные тесты для программы или ее фрагмента пишутся **до** самой программы (англ. *test-firstdevelopment*) и, по существу, управляют ее разработкой. Является одной из основных практик экстремального программирования.

Разработка в стиле TDD состоит из коротких циклов (длительностью от 2 мин в зависимости от опытности и стиля работы программиста). Каждый цикл включает следующие шаги.

1 Из репозитория извлекается программная система, находящаяся в согласованном состоянии, когда весь набор модульных тестов выполняется успешно.

2 Добавляется новый тест. Он может состоять в проверке, реализует ли система некоторое новое поведение или содержит ли некоторую ошибку, о которой недавно стало известно.

3 Успешно выполняется весь набор тестов, кроме нового теста, который выполняется неуспешно. Этот шаг необходим для проверки самого теста – включен ли он в общую систему тестирования и правильно ли отражает новое требование к системе, которому она, естественно, еще не удовлетворяет.

4 Программа изменяется с тем, чтобы **как можно скорее** выполнялись все тесты. Нужно добавить самое простое решение, удовлетворяющее новому тесту, и одновременно с этим не испортить существующие тесты. Большая часть нежелательных побочных и отдаленных эффектов от вносимых в программу изменений отслеживается именно на этом этапе с помощью достаточно полного набора тестов.

5 Весь набор тестов выполняется успешно.

6 Теперь, когда требуемая в этом цикле функциональность достигнута самым простым способом, программа перестраивается (рефакторинг) для улучшения структуры и устранения избыточного, дублированного кода.

7 Весь набор тестов выполняется успешно.

8 Комплект изменений, сделанных в этом цикле в тестах и программе, заносится в репозиторий (операция commit), после чего программа снова находится в согласованном состоянии и содержит четко осязаемое улучшение по сравнению с предыдущим состоянием.

Данный цикл упрощенно описывается Кентом Беком в своей книге как «красный – зеленый – рефакторинг». Красный и зеленый – это цвета полоски в среде тестирования JUnit, которая показывает, все тесты сработали или нет.

При этом на первом («красном») этапе необходимо добиться того, чтобы программа просто компилировалась, без срабатывания добавленного теста.

Задания (по вариантам)

Применяя методику разработки на основе тестирования (testdrivendesign, сначала тесты), разработать библиотеку классов для вариантов заданий, приведенных в таблице 1.

Разработка должна выполняться последовательно, в строгом соответствии с методикой. Для написания и выполнения тестов использовать ПО Unit (xUnit) для выбранного языка программирования.

Таблица 1 – Варианты заданий

Вариант	Задание
1	Дан текст. Сделать заглавной каждую букву каждого слова, начинающегося с заглавной буквы
2	Дан текст. В каждом слове текста заменить заданную литеру заданной литерой (сочетанием литер). Пример: Заменяемая литера: “б”, заменяющее сочетание литер: “ку”, слово: “а***б***ракада***б***ра”, результат: “а***ку***ракада***ку***ра”
3	В каждом слове удалить литеру, стоящую между двумя заданными.
4	Сформировать список, информирующий о вхождении заданной литеры в текст в виде ((<0 1 5 2 0>) (<3 0 1 5 2 0 1 0>)...). Цифры указывают количество вхождений литеры в каждое слово предложения
5	Дан текст. Заменить в каждом предложении все вхождения заданного слова на заданное новое слово
6	Дан текст. Удалить из каждого слова в каждом предложении все повторяющиеся литеры
7	Дан текст. В каждом слове каждого предложения для повторяющихся литер произвести следующую замену: повторные вхождения литер удалить, к первому вхождению литеры приписать число вхождений литеры в слово. Пример : ‘((aaabbccccddd)(eeefgggghhkl)) преобразуется в ‘((a3b2 c4d3)(e3fg3 h2kl))
8	Дан текст. В каждом слове вставить после заданного 3-буквенного сочетания заданное 2-буквенное
9	Дан текст. Вставить заданное новое слово после каждого вхождения другого заданного слова
10	Дан текст. Записать каждое предложение текста в порядке возрастания количества гласных букв в слове
11	Дан текст. Переписать каждое предложение, расположив слова в обратном алфавитном порядке
12	Написать программу, которая в каждом слове исходного текста меняет местами первую и последнюю буквы

Контрольные вопросы

1 Что такое разработка через тестирование?
2 Расшифруйте термин TDD.
3 Опишите кратко этапы цикла разработки TDD.

2 Лабораторная работа № 2. Экстремальное программирование. Scrum

Цель работы: изучить особенности разработки программ и роли на проекте по методологии Scrum. Получить практический опыт разработки ПО в команде по методологии Scrum.

2.1 Краткие теоретические сведения

Разработка программного обеспечения начинается с идеи: «Если бы у меня было некое ПО, которое выполняло бы примерно вот такие задачи!» Человека, который в команде будет представлять эту идею, называют ProductOwner (PO) или Владелец продукта. ProductOwner – это тот, кто видит цель продукта или кому кажется, что он видит цель продукта, но важно то, что он может ее сформулировать, донести эту цель до остальных участников команды. Цель он формирует в виде списка своих пожеланий – этот список называется ProductBacklogItems (PBI). Список постоянно модифицируется в зависимости от ситуации на рынке или от изменения требований ProductOwner'a. По Scrum считается, что наибольшая эффективность разработки достигается в том случае, если команда сама будет самостоятельно принимать решения в отношении того, как она будет двигаться к цели. Поэтому основное требование к команде – она должна быть самоорганизующейся и самоуправляемой. Так как требование серьезное, то в команду вводится роль ScrumMaster’a, который следит за тем, чтобы соблюдались правила Scrum.

Задание

Разработать программное обеспечение в соответствии с вариантом, используя методологию Scrum.

Студенты делятся на группы по три человека: один ScrumMaster и два разработчика.

В первый час пары студенты должны провести стартовый митинг и выбрать задачи для первого спринта (оставшаяся часть пары + доработка дома). Во время митинга требуется выполнить следующие задачи:

– разбить задачу по разработке ПО на более мелкие подзадачи;

– выбрать из ProductBacklogItems Вашего варианта наиболее приоритетные задачи (выбирает ScrumMaster);

– сделать оценку временных затрат на выполнение задач первого спринта;

– создать документ за результатами проведения спринта (пул задач, примерное время выполнения, кто выполняет). Ответственный – ScrumMaster.

После того, как разработка завершилась, требуется добавить в документ с результатами первого митинга реальное время, которое Вы потратили на выполнение задач.

По окончании спринта Вы будете предоставлять преподавателю результаты проведения спринта – документ, который Вы составляли, и презентацию

возможностей Вашего ПО. Преподаватель будет проверять соответствие возможностей ПО и требований в ProductBacklogItems. Также ScrumMaster должен отчитаться преподавателю о трудностях, которые возникли во время разработки, и о результатах спринта (совпало ли примерное время разработки с реальным и почему получился такой результат).

Рекомендации: можно менять ScrumMaster на каждой паре.

Контрольные вопросы

1 Какие роли есть в Scrum?
2 Что такое спринт и зачем он нужен?
3 Что происходит во время спринта?
4 Что происходит в конце спринта?

3 Лабораторная работа № 3. Анализ предметной области и требования к ПО

Цель работы: произвести анализ предметной области и разработку интерфейса информационной системы.

3.1 Краткие теоретические сведения

Функциональная спецификация – формальное описание, которое объясняет, что и как будет делать программа. Она достаточно детально показывает строение всех модулей и их взаимодействие с учетом проектных ограничений. Спецификация невозможна без четкого описания структур данных программы.

Функциональная спецификация разрабатывается под руководством и при непосредственном участии архитектора (бизнес-аналитика) проекта.

Функциональная спецификация может сильно изменяться от проекта к проекту. В крупных комплексных проектах спецификации имеют несколько уровней детализации. Первоисточником для разработки функциональных спецификаций является ***Техническое задание*** **(*CustomerRequirementsSpecification*)**. В этом документе описываются требования к программному продукту.

На *верхнем уровне* спецификирования программного продукта разрабатывается ***Внешняя спецификация*** для заказчика **(*SoftwareSpecificationDocument*)**. Это документ, объясняющий в бизнес-терминах, что должна делать система. Документ разрабатывается с ориентацией на пользователя и, соответственно, должен отражать все его интересы. Он не должен быть перегружен техническими подробностями. Пользователю интересно, какие меню, экраны и отчеты будут представлены в программе и как программа будет осуществлять переходы из одной точки в другую. В зависимости от уровня подготовленности заказчика спецификация может иметь различную степень детализации.

На *втором уровне* может быть разработан концептуальный документ,

описывающий ***Архитектуру системы*** **(*SoftwareArchitectureDocument, heightleveldesign*)**. Эта спецификация показывает функционирование всей системы в целом, не детализируя устройство отдельных модулей. Она представляет структуру объектов и их зависимости. Спецификация дает профессионалу быстрое понимание организации системы и функционирования компонент как части общей системы. Инструментами для описания архитектуры являются такие средства, как UML, DFD, ERD и т. п.

На заключительном этапе разрабатывается ***Техническая спецификация*** **(*DetailDesignDocument, lowleveldesign*)**. Этот документ является завершающим в цепочке спецификаций и позволяет полностью сосредоточиться на стадии кодирования системы. Спецификация описывает низкоуровневую организацию продукта. Здесь для каждого модуля разработчик должен определить все требования, включая передаваемые параметры, глобальные структуры и переменные, вызываемые подпрограммы и т. д. Эта информация важна для кодировщиков, которые параллельно реализуют различные модули, взаимодействующие друг с другом. Хорошо выполненная техническая спецификация снимает все проблемы, возникающие при объединении отдельных модулей в единое целое.

Порядок выполнения работы

1 Выполнить сбор сведений по проектируемой системе и анализ предметной области.

2 Разработать функциональные и нефункциональные требования к системе.

3 Оформить документ описания системы по шаблону Вигерса.

4 Изучить основные теоретические положения по разработке функциональной спецификации.

5 Спроектировать интерфейс информационной системы в соответствии с функциональными требованиями.

6 Сделать электронный отчет.

7 Защитить отчет у преподавателя.

Контрольные вопросы

1 Что такое функциональная спецификация и для чего она нужна?

2 Что является первоисточником для разработки функциональных спецификаций?

3 На каком этапе разрабатывается техническая спецификация?

4 Что разрабатывается на верхнем уровне специфицирования программного продукта?

4 Лабораторная работа № 4. Качество программного обеспечения

Цель работы: изучить методы оценки надежности и качества программного обеспечения АСОИУ.

4.1 Краткие теоретические сведения

Модели надежности программного и информационного обеспечения. Надежность является важным и естественным требованием, предъявляемым к качеству разрабатываемых программных и информационных компонентов АСОИУ. Теория надежности аппаратных средств АСОИУ разработана достаточно глубоко, поэтому при исследовании надежности программного и информационного обеспечения большинство методов используют идеи теории надежности технических средств. Это позволяет инженеру не только оценивать, но и прогнозировать надежность программно-информационных продуктов. С точки зрения надежности между техническими устройствами и программными продуктами, кроме несомненных аналогий, имеются существенные различия. Так, небрежно написанный и отлаженный программный модуль, непродуманная схема технического устройства могут давать неправильные результаты уже на первых этапах тестирования. Однако на этом сходство заканчивается. Дело в том, что любое, даже самое надежное, техническое устройство подвержено эксплуатационному износу и со временем начинает отказывать из-за старения. Программные же модули лишены подобных недостатков, и их надежность может в ходе эксплуатации только увеличиваться за счет устранения выявляемых ошибок. Поэтому теоретически может возникнуть ситуация, когда при тестировании программного модуля ошибки уже не будут обнаруживаться.

В процессе анализа надежности программное и информационное обеспечение (ИО) можно рассматривать как подсистемы АСОИУ.

Работоспособным называется такое состояние программного средства, при котором оно способно выполнять заданные функции с параметрами, установленными требованиями технического задания. С переходом в неработоспособное состояние связано событие отказа. Причиной отказа программного средства является невозможность его полной проверки в процессе тестирования и испытаний. При эксплуатации программного средства в реальных условиях может возникнуть такая комбинация входных данных, которая вызовет отказ. Следовательно, работоспособность программного средства зависит от входных данных, и чем меньше эта зависимость, тем выше уровень надежности.

В общем случае отказы программного обеспечения определяются как отклонения от правильного хода выполнения программы вследствие ошибок, допущенных в процессе преобразования исходного алгоритма в действующую программу. А **ошибка** – это регистрируемый пользователем факт неудовлетворенности качеством программы, причина дефекта системы программного обеспечения; и наоборот, дефект рассматривается как проявление допущен-

ной ранее ошибки.

Таким образом, надежность программного (и информационного) обеспечения можно определить как вероятность того, что отказ программного обеспечения, вызывающий отклонение получаемых результатов – выход их за допустимые пределы, не произойдет в течение заданного периода времени (при определенных условиях внешней среды).

Следует иметь в виду, что не все отказы приводят к уменьшению надежности программного обеспечения, а только те, которые вызывают отклонение результата расчета от допустимых пределов.

Модель надежности программного и информационного обеспечения – это математическая модель, построенная для оценки зависимости надежности программного и информационного обеспечения от некоторых определенных параметров. Например, параметров, связанных с какой-либо ветвью программы на подмножестве наборов входных данных, с помощью которых эта ветвь контролируется. Или частоты ошибок, которая позволяет оценить качество систем реального времени, функционирующих в непрерывном режиме, и при этом получить косвенные данные о надежности программного и информационного обеспечения.

Известные модели надежности программного и информационного обеспечения АСОИУ можно классифицировать по различным признакам, в частности по тому, какой из перечисленных процессов они поддерживают (предсказывающие, прогнозные, оценивающие, измеряющие). Модели надежности, которые в качестве исходной информации используют данные об интервалах между отказами, можно отнести к измеряющим или к оценивающим в равной степени. Некоторые модели, основанные на информации, полученной в ходе тестирования программного средства, дают возможность делать прогнозы его поведения в процессе эксплуатации.

Аналитические модели позволяют рассчитать количественные показатели надежности, основываясь на данных о поведении программы в процессе тестирования (измеряющие и оценивающие модели).

Аналитические модели реализуются в виде **динамических и статических**. В динамических моделях надежности программно-информационных средств поведение программы (появление отказов) рассматривается во времени. В статических моделях появление отказов не связывают со временем, а учитывают только зависимость количества ошибок от числа тестовых прогонов (по области ошибок) или зависимость количества ошибок от характеристики входных данных (по области данных). Именно аналитическим моделям уделено особое внимание.

Эмпирические модели базируются на анализе структурных особенностей программ. При разработке эмпирических моделей надежности программного и информационного обеспечения предполагается, что связь между надежностью и другими параметрами является статической. Эти модели рассматривают зависимость показателей надежности от числа межмодульных связей, количества циклов в модулях, отношения количества прямолинейных участков к

количеству точек ветвления и т. п. Иначе говоря, при разработке эмпирической модели стремятся иметь дело с такими параметрами, соответствующее изменение значений которых должно приводить к повышению надежности программного и информационного обеспечения. Необходимо отметить, что нередко эмпирические модели не дают конечных результатов показателей надежности.

Основным средством определения количественных показателей надежности являются **модели надежности**, под которыми понимают математические модели, построенные для оценки зависимости надежности от заранее известных или оцененных в ходе создания программного средства параметров. В связи с этим определение надежности показателей принято рассматривать в единстве трех процессов – предсказание. измерение, оценивание.

Задание

Получить показатели надежности заданного программного модуля на основе динамической, статической и эмпирической моделей.

Контрольные вопросы

1 Что называется работоспособным состоянием ПО?

2 Как можно определить надежность программного (информационного) обеспечения?

3 Что такое модель надежности программного и информационного обеспечения?

4 На чём базируются эмпирические модели?

5 Лабораторная работа № 5. WPF. Разработка корпоративного приложения. Интерфейс

Цель работы: ознакомиться с WPF-приложениями и структурой этих приложений.

5.1 Краткие теоретические сведения

Для разработки приложения необходимо создать WPF-проект, инструментальная система сгенерирует следующий XAML-документ:

```
<Window x:Class="WpfApplProject.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WpfApplProject"
mc:Ignorable="d"
Title="ИнформационнаясистемаВУ" Height="450" Width="825">
```

```
    <Grid>
        <Frame x:Name="frame" Content="Frame" Margin="3"
    Source="/WpfApplProject;component/PageMain.xaml" NavigationUIVisibility="Hidden">
            <Frame.Background>
                <LinearGradientBrush>
                    <LinearGradientBrush.GradientStops>
                        <GradientStop Offset="0.00" Color="#FFFF8686"/>
                        <GradientStop Offset="1.00" Color="#FFA5B1FF"/>
                    </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>

            </Frame.Background>
        </Frame>
    </Grid>
</Window>
```

В данном XAML-документе имеется один элемент верхнего уровня <Window>. Дескриптор </Window> завершает весь документ. В XAML-документе приведено имя класса MainWindow:

x:Class="WpfApplProject.MainWindow

Два пространства имен:

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml

Три свойства:

Title="MainWindow" Height="350" Width="525"

Каждый атрибут соответствует определенному свойству класса Window. Приведенные атрибуты предписывают WPF создать окно с надписью MainWindow и размером 350 × 525 единиц.

Когда выполняется компиляция приложения, XAML-файл, который определяет пользовательский интерфейс (MainWindow.xaml), транслируется в объявление типа CLR, которое объединяется с логикой приложения из файла класса отдельного кода (MainWindow.xaml.cs).

Метод InitializeComponent() генерируется во время компиляции приложения и в исходном коде не присутствует.

Для программного управления элементами управления, описанными в XAML-документе, нужно задать XAML атрибут Name. Так, для задания имени элементу Grid необходимо записать следующую разметку:

```
<GridName="grid">

</Grid>
```

Страницы приложения можно размещать внутри окон и внутри других страниц. В WPF при создании страничных приложений контейнером наивысшего уровня могут быть следующие объекты:

– NavigationWindow, который представляет собой несколько видоизмененную версию класса Window;

– Frame, находящийся внутри другого окна или другой страницы;

– Frame, обслуживаемый непосредственно в Internetexplorer.

5.2 Варианты заданий

Создать интерфейс WPF-приложения с навигацией по страницам для работы с базой данных. Разрабатываемое приложение должно обеспечивать хранение и обработку указанных данных по варианту. Функции приложения:

– просмотр данных;

– ввод данных;

– редактирование данных;

– удаление данных;

– поиск данных по клиенту.

Вариант 1. Информационная подсистема ведения счетов клиентов.

БД включает следующие таблицы: Account, Aggreement, Bank, TypeAccount.

Назначение подсистемы ведения счетов клиентов – поддержание в актуальном состоянии инвестиционных счетов клиентов.

Назначение атрибутов таблицы Счет – Account:

- ID – суррогатный ключ;
- TypeID – внешний ключ для связи с таблицей TypeAccount;
- BankID – внешний ключ для связи с таблицей Bank;
- AgreementID – внешний ключ для связи с таблицей Agreement;
- Account – номер инвестиционного счета.

Назначение атрибутов таблицы Тип счета – TypeAccount:

- ID – суррогатный ключ;
- TypeAccount – тип счета.

Назначение атрибутов таблицы Банк – Bank:

- ID – суррогатный ключ;
- NameFull – полное наименование банка;
- NameShort – краткое наименование банка;
- Inn – ИНН банка;
- Bik – БИК банка;
- CorAccount – номер корсчета;
- Account – номер счета;
- City – город.

Назначение атрибутов таблицы Договор – Agreement:

- ID – суррогатный ключ;
- PersonID – внешний ключ для связи с таблицей Person;
- TypeID – внешний ключ для связи с таблицей Type;

- StatusID – внешний ключ для связи с таблицей Status;
- Number – номер договора;
- DataOpen – дата заключения договора;
- DataClouse – дата закрытия договора;
- Note – пояснения.

Вариант 2. Информационная подсистема ведения адресов клиентов.

БД включает следующие таблицы: City, Region, Address, Country.

Назначение подсистемы ведения адресов клиентов – поддержание в актуальном состоянии адресов клиентов.

Назначение атрибутов таблицы Адрес – Address:

- ID – суррогатный ключ;
- IndexAddress – адресный индекс;
- PersonID – внешний ключ для связи с таблицей Person;
- CountryID – внешний ключ для связи с таблицей Country;
- RegionID – внешний ключ для связи с таблицей Region;
- CityID – внешний ключ для связи с таблицей City;
- Street – наименование улицы;
- Bulding – номер строения, дома;
- Office – номер офиса.

Назначение атрибутов таблицы Город – City:

- ID – суррогатный ключ;
- RegionID – внешний ключ для связи с таблицей Region;
- CountryID – внешний ключ для связи с таблицей Country;
- City – город.

Назначение атрибутов таблицы Регион – Region:

- ID – суррогатный ключ;
- CountryID – внешний ключ для связи с таблицей Country;
- Region – регион.
- Назначение атрибутов таблицы Страна – Country:
- ID – суррогатный ключ;
- CountryFull – полное наименование страны;
- CountryShort – краткое наименование страны.

Вариант 3. Информационная подсистема ведения договоров клиентов.

БД включает следующие таблицы: Person, Aggreement, StatusAggrement, TypeAggrement.

Назначение подсистемы ведения договоров клиентов – поддержание в актуальном состоянии договоров клиентов.

Назначение атрибутов таблицы Договор – Agreement:

- ID – суррогатный ключ;
- PersonID – внешний ключ для связи с таблицей Person;
- TypeID – внешний ключ для связи с таблицей Type;
- StatusID – внешний ключ для связи с таблицей Status;

- Number – номер договора;
- DataOpen – дата заключения договора;
- DataClouse – дата закрытия договора;
- Note – пояснения.

Назначение атрибутов таблицы Статус договор – StatusAggrement:

- ID – суррогатный ключ;
- Status – статус договора.

Назначение атрибутов таблицы Клиент – Person:

- ID – суррогатный ключ;
- OrgLicenseID – внешний ключ для связи с таблицей OrgLicense;
- VerietyID – внешний ключ для связи с таблицей Veriety;
- StatusID – внешний ключ для связи с таблицей Status;
- Inn – ИНН клиента;
- Type – тип клиента;
- Shifer – шифр клиента;
- Data – дата регистрации клиента.

Вариант 4. Информационная подсистема ведения клиентов – физических лиц.

БД включает следующие таблицы: Person, Citizen, Document.

Назначение подсистемы ведения клиентов – поддержание в актуальном состоянии информации по клиентам – физическим лицам.

Назначение атрибутов таблицы Клиент – Person:

- ID – суррогатный ключ;
- OrgLicenseID – внешний ключ для связи с таблицей OrgLicense;
- VerietyID – внешний ключ для связи с таблицей Veriety;
- StatusID – внешний ключ для связи с таблицей Status;
- Inn – ИНН клиента;
- Type – тип клиента;
- Shifer – шифр клиента;
- Data – дата регистрации клиента.

Назначение атрибутов таблицы Физическое лицо – Citizen:

- ID – суррогатный ключ;
- DocumentID – внешний ключ для связи с таблицей Document;
- SurName – фамилия клиента;
- Name – имя клиента;
- Patronic – отчество клиента;
- Number – номер документа, удостоверяющего личность;
- Seriy – серия документа, удостоверяющего личность;
- Organ – орган, выдавший документ, удостоверяющий личность;
- Data – дата выдачи документа, удостоверяющего личность.

Назначение атрибутов таблицы Документ – Document:

- ID – суррогатный ключ;
- Document – наименование документа, удостоверяющего личность.

Контрольные вопросы

1 Какой элемент верхнего уровня есть в XAML-документе, который генерирует система при создании WPF-приложения?
2 Как задать имя элементу в XAML-документе?
3 В каких контейнерах можно размещать страницы WPF?
4 Из каких объектов можно сформировать меню приложения?
5 Какие элементы контроля используются для перехода между страницами приложения?

6 Лабораторная работа № 6. WPF. Разработка корпоративного приложения. Подключение к БД

Цель работы: изучить способы подключения к БД.

6.1 Краткие теоретические сведения

Для взаимодействия приложения с базой данных необходимо в коде класса страницы объявить статическое свойство контекста данных сформированной EDM-модели. Это свойство целесообразно объявлять статическим. Например:

```
publicstaticTitlePresonalEntitiesDataEntitiesEmployee {get; set;}
```

Также необходимо объявить обобщенную коллекцию типа ObservableCollection<Employee> для работы приложения с коллекцией объектов базы данных. Этот тип представляет коллекцию динамических данных, обеспечивающих выдачу уведомления при получении и удалении элементов или при обновлении всего списка. Тип ObservableCollection<T> находится в пространстве имен System.Collections.ObjectModel, ссылку на которое нужно добавить в объявлении класса приложения.

```
usingSystem.Collections.ObjectModel;
```

Экземпляры свойств контекста данных и коллекции необходимо создать в конструкторе класса страницы. Например:

```
public partial class PageEmployee : Page
{
public static TitleEmployeeEntities Context = new TitleEmployeeEntities();
publicPageEmployee()
{
InitializeComponent();
}
…
}
```

Формирование данных для приложения, которые должны предоставляться из базы данных, будет проводиться при загрузке страницы приложения. Для этого в XAML-документ Page добавьте свойство Loaded.

```
Loaded="Page_Loaded"
```

В код класса страницы приложения включаем обработчик Page_Loaded.

```
private void filter()
{
var query = from e in Context.Employe
where e.Surname.Contains(TextBoxSurname.Text)
select e;
if(ComboBoxTitle.SelectedIndex> -1)
{
query = from e in query
where e.ID_tittle == ((Title)ComboBoxTitle.SelectedValue).ID
select e;
}
DataGridEmployee.ItemsSource = query.ToList();
}
```

Поле employees имеет тип ObjectQuery<Employee>. Класс ObjectQuery<T> представляет запрос, возвращающий коллекцию типизированных сущностей с любым количеством элементов. Запрос сформируем с помощью технологии LINQ.

```
varquery = fromeinContext.Employe
where e.Surname.Contains(TextBoxSurname.Text)
selecte;
```

Результаты запроса целесообразно отсортировать, например, по фамилии сотрудника (orderbyemployee.Surname). Далее формируем коллекцию объектов базы данных и источник данных для сетки DataGrid.

```
foreach (Employee emp in queryEmployee)
{
ListEmployee.Add(emp);
}
DataGridEmployee.ItemsSource = ListEmployee;
```

В результате проектирования в приложении сформирована коллекция с данными из таблицы базы данных.

Необходимо настроить сетку DataGrid для корректного отображения данных. Модифицируйте общее описание DataGrid.

1 Определите привязку для источника данных.

```
ItemsSource="{Binding}"
```

2 Отмените автоматическую генерацию столбцов.

```
AutoGenerateColumns="False"
```

3 Установите привязку к левому краю страницы.

```
HorizontalAlignment="Left"
```

4 Определите максимальные размеры сетки.

```
MaxWidth="1000" MaxHeight="295"
```

5 Установите основной и альтернативный цвета заливки сетки.

```
RowBackground="#FFE6D3EF"
AlternatingRowBackground="#FC96CFD4"
```

6 Определите цвет заливки и толщину линии для рамки сетки.

```
BorderBrush="#FF1F33EB"
BorderThickness="3"
```

7 Определите высоту строк сетки.

```
RowHeight="25"
```

8 Переопределите форму курсора при наведении указателя мыши на таблицу DataGridEmployee.

Модифицированное XAML-описание сетки DataGrid:

```
<DataGrid Name="DataGridEmployee"
AutoGenerateColumns="False"
HorizontalAlignment="Left"
RowBackground="#FFE6D3EF"
AlternatingRowBackground="#FC96CFD4"
BorderBrush="#FF1F33EB"
BorderThickness="3"
RowHeight="25"
Cursor="Hand"
IsReadOnly="True"
Width="Auto">
```


Привязка текстового поля. Для каждого текстового столбца задайте свойство привязки Binding. В расширении разметки привязки данного свойства определите свойство класса источника данных, к которому привязывается колонка. Далее укажите режим двусторонней привязки (Mode=TwoWay), который определяет синхронное обновление как источника, так и приемника привязки. Способ обновления источника привязки задается свойством UpdateSourceTrigger, которое принимает значение PropertyChanged, что соответствует немедленному обновлению источника привязки каждый раз при изменении свойства цели привязки. Например:

```
<DataGridTextColumn Header="Фамилия" Binding="{Binding Surname, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>
```

Привязка выпадающего списка. Привязка данных к колонке типа DataGridComboBoxColumn требует определенной подготовительной работы. Если в таблице модели данных хранится не текстовое значение поля, а внешний

ключ для другой таблицы, где находятся данные, то в EDM-модели можно получить значение атрибута из связанных таблиц. Для этого используется атрибут связи в таблице EDM-модели.

Например, для обеспечения возможности работы с коллекцией таблицы Title в приложении добавьте в проект папку Model и в ней создайте класс ListTitle.

```
public class ListTitle :ObservableCollection<Title>
{
public ListTitle()
{
TitleEmployeeEntities context = new TitleEmployeeEntities();
context.Title.Load();
foreach (Title titl in context.Title.Local)
{
Add(titl);}}}
```

Класс ListTitle наследуется от класса обобщенной коллекции ObservableCollection<T> и его назначение – создавать коллекцию объектов Title. Поле titles является запросом типа ObjectQuery<Title>. Данному полю присваивается свойство Titles контекста данных DataEntitiesEmployee, который определен в классе приложения.

Запрос LINQ получает данные из базы данных в поле queryTitle и затем в цикле foreach формируется коллекция класса ListTitle.

Для использования класса ListTitle в XAML-документе класса приложения необходимо объявить данный класс как ресурс. При этом нужно подключить пространство имен WpfApplProject.Model.

```
xmlns:e ="clr-namespace:WpfApplProject.Model"
```

Затем определите ресурс страницы с ключом listTitle.

```
<Page.Resources>
<e:ListTitle x:Key="ListTitles"/>
</Page.Resources>
```

Далее модифицируйте XAML-описание для типа DataGridComboBoxColumn.

```
<DataGridComboBoxColumn Header="Должность"
ItemsSource="{Binding Source={StaticResourceListTitles}}"
DisplayMemberPath="title1"
SelectedValueBinding="{Binding Path=ID_tittle, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedValuePath="ID" />
```

Источник выпадающего списка задается как статический ресурс {StaticResourcelistTitle}. Выводимое в ячейки колонки поле должно соответствовать полю Title1 таблицы Title EDM-модели

(DisplayMemberPath="Title1"). Выбираемый в списке параметр (SelectedValueBinding) должен быть привязан к полю TitleID таблицы Employee.

```
SelectedValueBinding="{Binding Path=TitleID, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
```

Выбор свойства SelectedValueBinding производится по пути, определенному свойством SelectedValuePath (SelectedValuePath="ID").

Привязка даты. При привязке даты ставится задача для невыделенной ячейки представлять дату в виде цифр дня, месяца и года, разделенных точками, а для выделенной ячейки – использовать класс для выбора даты.

Решение данной задачи осуществлено с помощью разработки двух шаблонов данных DataTemplate.

В первом шаблоне, для которого задан ключ DateTemplate, используется элемент управления TextBlock, свойство Text которого привязывается к атрибуту BirstDate таблицы Employee. Данные в привязке форматируются свойством StringFormat и строковые данные выравниваются по центру. Этот шаблон применяется для визуализации данных в режиме их просмотра.

```
<DataTemplatex:Key="DateTemplate" >
<TextBlock Text="{Binding BirthData, StringFormat={}{0:dd\.}{0:MM\.}{0:yyyy}}"
VerticalAlignment="Center" HorizontalAlignment="Center" />
</DataTemplate>
```

Второй шаблон с ключом EditingDateTemplate использует класс DatePicker. Этот шаблон применяется в режиме редактирования данных.

```
<DataTemplatex:Key="EditingDateTemplate">
<DatePickerSelectedDate="{Binding BirthData, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
</DataTemplate>
```

Разработанные шаблоны необходимо добавить к ресурсам в XAML-документ страницы PageEmployee.

6.2 Варианты заданий

Разработайте базу данных согласно вариантам подразд. 5.2 и подключите ее к приложению, созданному в лабораторной работе № 5.

Контрольные вопросы

1 Что необходимо объявить в коде класса страницы для взаимодействия приложения с БД?

2 Что такое Binding?

3 Что Вы знаете о классе ListTitle?

4 Для чего предназначена технология LINQ?

7 Лабораторная работа № 7. Разработка приложений ASP.NET по шаблону MVC

Цель работы: ознакомиться с шаблоном MVC и получить опыт разработки приложения.

Схема архитектуры **Model-View-Controller (MVC)** разделяет приложение на три основных компонента: **модель**, **представление** и **контроллер**. Платформа ASP.NET MVC представляет собой альтернативу схеме веб-форм ASP.NET при создании веб-приложений. Платформа ASP.NET MVC является легковесной платформой отображения с широкими возможностями тестирования и, подобно приложениям на основе веб-форм, интегрирована с существующими функциями ASP.NET, например, с главными страницами и проверкой подлинности на основе членства. Платформа MVC определяется в сборке **System.Web.Mvc**.

Задание

Создание простого MVC 5 приложения ASP.NET в VisualStudio 2015.

В меню Файл выберите Создать –> Проект. Откроется диалоговое окно Создать проект. Выберите вкладку Web в левой части окна, далее – проект Asp.NET Web Application.

Откроется диалоговое окно выбора типа проекта. Необходимо выбрать MVC и настроить параметры аутентификации пользователей и другие предлагаемые настройки. Далее нажмите ОК (рисунок 7.1).

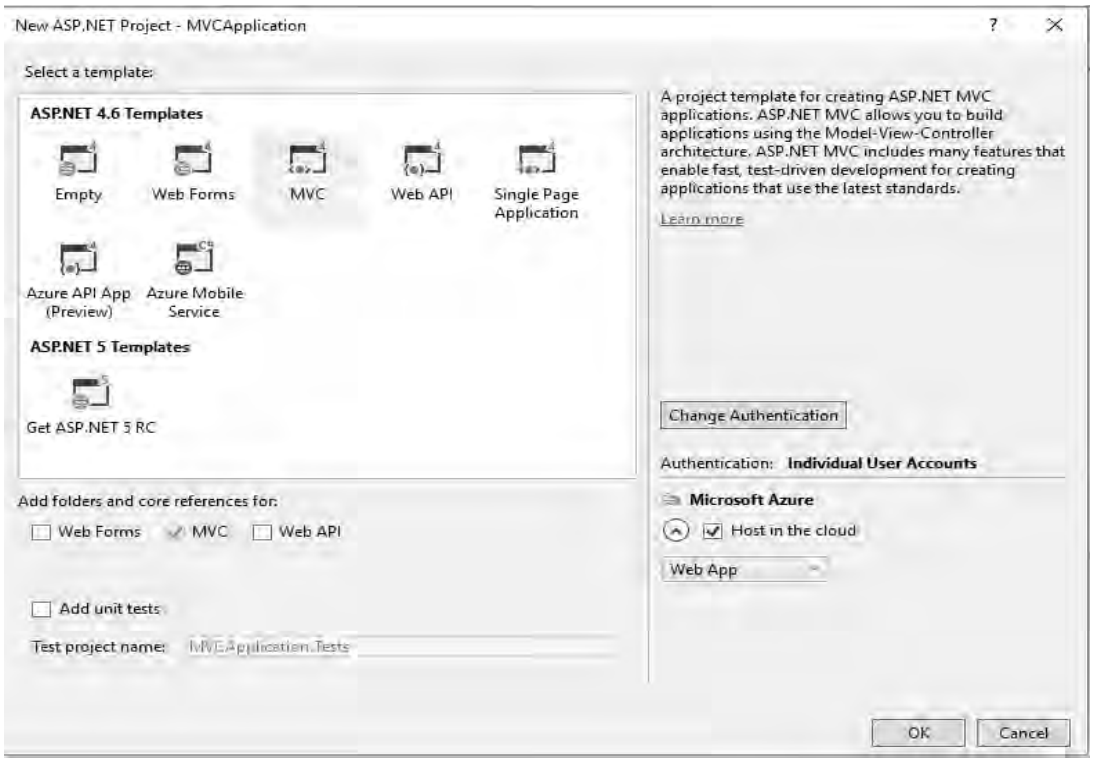

Рисунок 7.1 – Настройка параметров создаваемого проекта

Добавьте **NewItem** для добавления нового источника данных (рисунок 7.2).

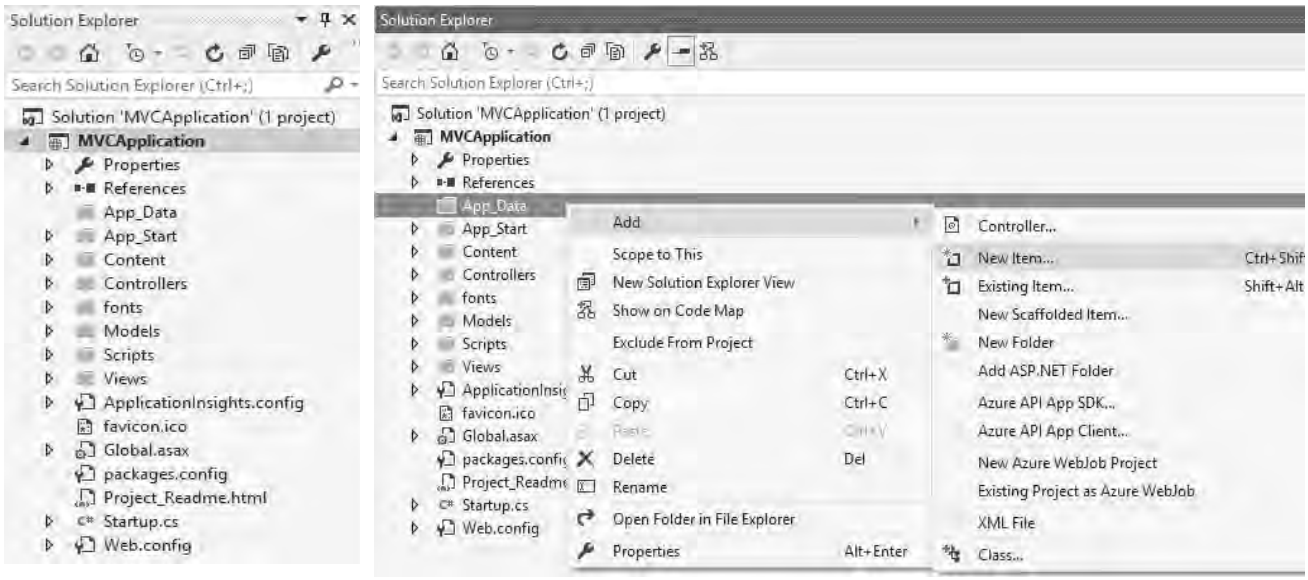

Рисунок 7.2 – Процесс добавления нового элемента в проект

После этого добавьте локальную базу данных в разделе Data.

Откройте базу для создания таблиц, после чего добавьте новую таблицу в базу (рисунок 7.3).

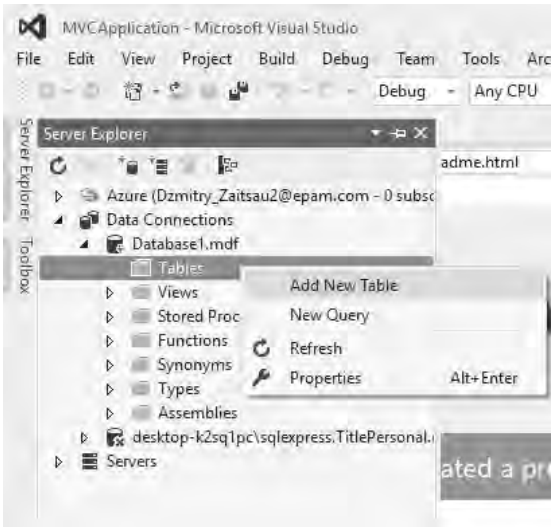

Рисунок 7.3 – Процесс добавления новой таблицы в базу данных

Добавьте поле **id**. Внизу в свойствах поля в **IdentitySpecification** поставьте параметр **IsIdentity** в **Yes**. Таким образом, получится поле **id** с автоинкрементом с шагом 1. В параметре **Identityincrement** можно выставить любой другой шаг. Шаг должен быть целым числом больше 0. Далее добавьте еще несколько полей: **Имя (Name)** и **Должность (Post)**. Затем установите **Primarykey** полю **id**. Результат показан на рисунке 7.4.

Далее требуется разработать MVC приложение для работы с базой данных,

которое содержит представления для просмотра, добавления, редактирования и удаления информации о сотрудниках, а также все необходимые компоненты.

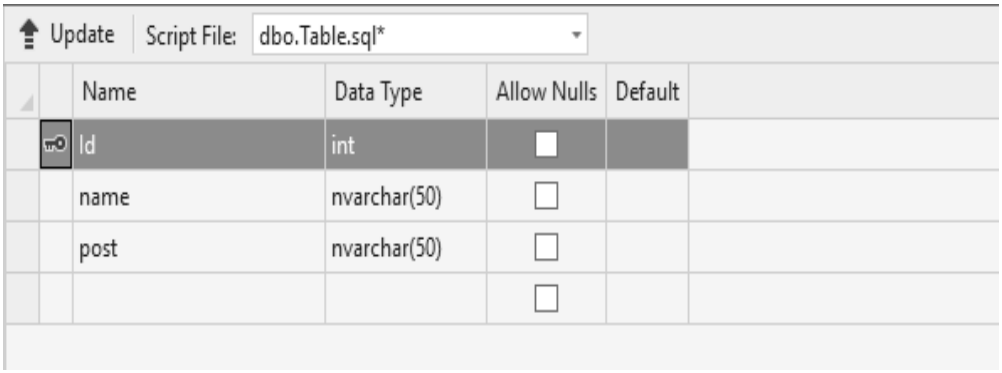

Рисунок 7.4 – Сконфигурированная таблица базы данных

Контрольные вопросы

1 Назовите три основных компонента MVC и охарактеризуйте их.
2 Каким образом схема MVC упрощает тестирование?
3 Назовите преимущества платформы ASP.NET MVC.

8 Лабораторная работа № 8. Разработка мобильных приложений для ОС Андроид (комплексная)

Цель работы: разработать простое приложение, помогающее понять структуру приложения для ОС Андроид, освоить основные операторы, привыкнуть к среде разработки.

Задачи лабораторной работы:
– создать новое приложение и изучить его структуру;
– настроить интерфейс приложения;
– реализовать логику приложения.

Для достижения поставленной цели в лабораторной работе создадим приложение в среде разработки AndroidIDE (Eclipse и ADT), подробно рассмотрим структуру полученного проекта и разберем назначение основных его элементов.

Сформулируем задачу, которую будет решать приложение, названное «Угадай число». Суть приложения в том, что программа случайным образом «загадывает» число от 0 до 100, а пользователь должен угадать это число. При каждом вводе числа программа сообщает пользователю результат: введенное число больше загаданного, меньше или же «Ура, победа!» число угадано.

Разрабатываемое приложение выполняет свои функции только тогда, когда видимо на экране. Когда оно не видимо, его работа приостанавливается,

т. е. имеем дело с приложением переднего плана. Для выполнения всей работы достаточно определить одну активность в приложении, фоновые процессы не предусмотрены.

Далее в работе рассмотрим простейшие элементы интерфейса пользователя и добавим их в приложение, а также рассмотрим вопросы, связанные непосредственно с программированием: научимся обрабатывать события, возникающие при взаимодействии приложения с пользователем; реализуем логику проверки числа на совпадение с загаданным.

8.1 Создание приложения и изучение его структуры

Создайте новый проект в среде AndroidIDE (Eclipse с ADT). В процессе создания проекта, названного ProjectN, среда разработки подготавливает необходимые папки и файлы. Полный иерархический список обязательных элементов проекта можно увидеть на вкладке PackageExplorer (аналогичную информацию предоставляет вкладка ProjectExplorer). Иерархия полученных папок и файлов для данного проекта изображена на рисунке 8.1.

Рисунок 8.1 – Структура проекта ProjectN

В настоящее время будет интересно назначение нескольких файлов и папок.

Рассмотрим папки:

– папка **src** – содержит файлы с исходным кодом на языке Java. Именно в этой папке размещаются все классы, создаваемые в процессе разработки приложения. Сейчас в данной папке в пакете com.example.projectn размещается единственный класс MainActivity.java, определяющий главную и единственную активность в этом приложении;

– папка **gen** – содержит java-файлы, которые не требуется изменять и лучше вообще не трогать. Эти файлы генерируются автоматически. Интересен файл R.java, содержащий идентификаторы (ID) для всех ресурсов приложения;

– папка **res** – содержит структуру папок ресурсов приложения. Рассмотрим некоторые из них:

– layout – в данной папке содержатся xml-файлы, которые описывают внешний вид форм и их элементов, пока там находится только activity_main.xml;

– values – содержит xml-файлы, которые определяют простые значения таких ресурсов, как строки, числа, цвета, темы, стили, которые можно использовать в данном проекте;

– menu – содержит xml-файлы, которые определяют все меню приложения.

8.2 Настройка интерфейса приложения

Пришло время задуматься о внешнем виде приложения. Для начала необходимо определить, какие элементы графического интерфейса нужны, как эти элементы будут располагаться на форме и каким образом будет реализовано взаимодействие с пользователем.

Так как приложение очень простое, то и интерфейс особой сложностью отличаться не будет. Требуются поле для ввода чисел (**TextEdit**), текстовая метка для вывода информации (**TextView**) и кнопка для подтверждения введенного числа (**Button**). Располагать элементы интерфейса будем друг под другом. Сверху – информационная часть, ниже – поле ввода, кнопку разместим в самом низу приложения. Взаимодействие приложения с пользователем организуется очень просто: пользователь вводит число в поле для ввода и нажимает кнопку, читает результат в информационном поле и либо радуется победе, либо вводит новое число.

Если пользователь вводит правильное число, то приложение предлагает ему сыграть снова, при этом кнопка будет играть роль подтверждения, а в информационное поле будет выведено приглашение к повторной игре.

Схематично интерфейс приложения изображен на рисунке 8.2.

Необходимо добавить на форму три элемента: информационное поле (**TextView**), поле ввода (**TextEdit**) и кнопку (**Button**). В результате получим интерфейс, изображенный на рисунке 8.3.

Задание

Разработать логику и программный код приложения. Результат приведен на рисунке 8.4, а пример кода – далее. Проверить работу приложения на мобильном устройстве.

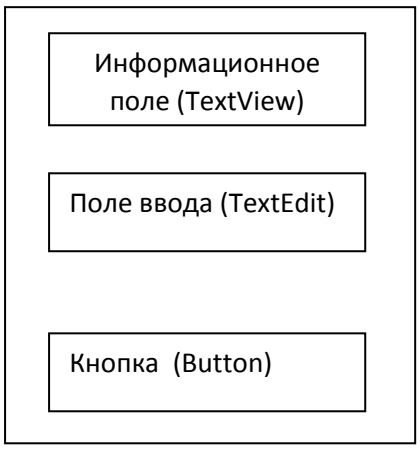

Рисунок 8.2 – Схема интерфейса приложения «Угадай число»

Рисунок 8.3 – Интерфейс приложения «Угадай число»

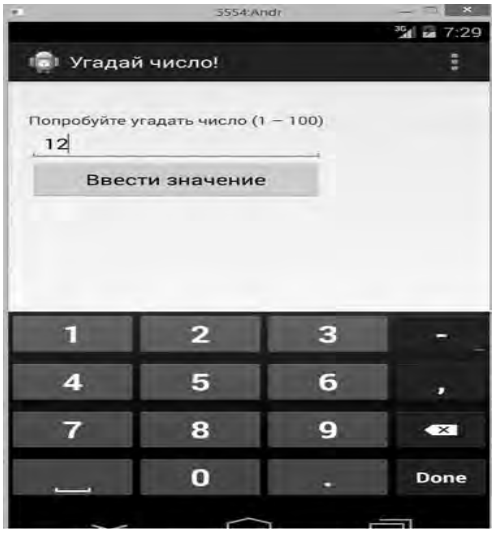

Рисунок 8.4 – Интерфейс приложения «Угадай число» при тестировании

Пример кода для мобильного устройства:

```
package com.example.projectn;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.*;

public class MainActivity extends Activity {
TextViewtvInfo;
EditTextetInput;
  Button bControl;

int guess;
booleangameFinished;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

tvInfo = (TextView)findViewById(R.id.textView1);
etInput = (EditText)findViewById(R.id.editText1);
bControl = (Button)findViewById(R.id.button1);

    guess = (int)(Math.random()*100);
gameFinished = false;
  }

  @Override
  public booleanonCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.main, menu);
    return true;
  }
  public void onClick(View v){
    if (!gameFinished){
intinp=Integer.parseInt(etInput.getText().toString());
      if (inp> guess)
tvInfo.setText(getResources().getString(R.string.ahead));
      if (inp< guess)
tvInfo.setText(getResources().getString(R.string.behind));
      if (inp == guess)
      {
tvInfo.setText(getResources().getString(R.string.hit));
bControl.setText(getResources().getString(R.string.play_more));
gameFinished = true;
      }
    }
    else
    {
      guess = (int)(Math.random()*100);
```

```
        bControl.setText(getResources().getString(R.string.input_value));
    tvInfo.setText(getResources().getString(R.string.try_to_guess));
    gameFinished = false;
       }
    etInput.setText("");
      }
    }
```

Контрольные вопросы

1 Как создать новый проект в среде AndroidIDE (Eclipse с ADT)?

2 Опишите основные файлы и папки проекта.

3 Какие элементы управления можно использовать для создания простых интерфейсов и как их добавить?

Список литературы

1 **Мак-Дональд, М.** WPF : Windows Presentation Foundation в .NET 3.5 с примерами на C# 2008 для профессионалов / М. Мак-Дональд. – 3-е изд. – Москва : И. Д. Вильямс, 2013. – 928 с.

2 **Петцольд, Ч.** Microsoft Windows Presentation Foundation. Базовый курс / Ч. Петцольд. – Санкт-Петербург : Питер, 2012 – 945 с.

3 **Дейтел, П.** Как программировать на C# 2012 / П. Дейтел, Х. Дейтел. – Санкт-Петербург : Питер, 2014. – 864 с.

4 **Эспозито, Д.** Microsoft ASP.NET 2.0. Углубленное изучение : пер. с англ. / Д. Эспозито. – Санкт-Петербург : Питер, 2012. – 592 с.

5 **Кулямин, В. В.** Технологии программирования. Компонентный подход : учебное пособие / В. В. Кулямин. – Москва : Интернет-Университет Информационных Технологий ; Бином ; Лаборатория знаний, 2016. – 463 с.

6 **Колесниченко, Д. Н.** Программирование для Андроид : самоучитель / Д. Н. Колесниченко. – Санкт-Петербург : Питер, 2014. – 242 с.