СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Данный курс нацелен на обучение студентов таким технологиям и принципам, которые являются на сегодняшний момент, наряду с ООП, неотъемлемой частью разработки крупных приложений. К таким технологиям относятся: системы контроля версий (Version Control System, VCS), позволяющие управлять изменениями в программном коде; системы отслеживания ошибок и задач (Bug tracking System, Issue Tracking System). К принципам относятся, принципы объектно-ориентированного проектирования (дизайна) (ООД), позволяющие эффективно организовывать программные структуры в разрабатываемом коде, а именно: отделение логики от представления, отделение данных от логики, слабое связывание (Low Coupling), высокое зацепление [за обязанности] (High Cohesion), шаблоны проектирования и другое.

Лабораторная работа № 1
Оформление текстового документа в Microsoft Office Word

Цель работы:

Изучение основных средств Microsoft Office Word, позволяющих существенно упростить и автоматизировать создание текстовых документов**.**

Введение

Умение работать с документацией является одним из важных навыков для профессионального разработчика программного обеспечения. Разработчику постоянно приходиться сталкиваться с проектной документацией: техническими заданиями, спецификацией требования к программному обеспечению и пр. Разработчик ПО должен обладать не только навыками написания кода, но и аналитическими навыками. При необходимости он должен уметь описать техническое решение и оформить его в виде документа. Данному навыку и посвящена первая лабораторная работа.

На сегодняшний день широко распространены различные офисные приложения, лидером среди них можно назвать линейку продуктов корпорации Microsoft. Важное место среди офисных приложений занимает текстовый редактор, являясь одним из основных инструментов в сфере документооборота. Большинство пользователей, однако, работает в основном с базовыми функциями ввода, редактирования и форматирования текста. Выполнение данной лабораторной работы позволит студенту изучить и освоить функции редактора Microsoft Word, которые существенно упростят и автоматизируют процесс создания документации.

Основная теоретическая часть

1.1 Создание документа в Microsoft Office Word

При запуске приложения будет создан новый текстовый документ. При открытии ранее созданных файлов, в зависимости от версии приложения, в котором они созданы, открываются в режиме полной (*.docx, *docm и т.д.) или ограниченной функциональности (*.doc). Общий вид приложения Microsoft Office Word 2010 представлен на рис. 1.1.

1.2 Форматирование текста

Форматирование текста в Word 2010 осуществляется тремя способами: форматирование отдельных символов, форматирование абзацев и использование стилей. Первые два способа подразумевают изменение параметров выделенного текста по желанию пользователя – выбор шрифта, его размера, написания и цвета, типа выравнивания абзаца на странице и так далее. Использование одного из стилей позволяет присвоить тексту уже готовый набор параметров, изменив также его структуру – обычный текст может превратиться в заголовок, подзаголовок, цитату и так далее.

Стили

Более удобным при работе с объемными документами является возможность использования стандартных стилей. Стилями называются наборы определенных параметров форматирования, соответствующие фрагментам текста разного уровня: заголовок, подзаголовок, цитата, ссылка, основной текст и так далее. Применяя один стиль к разным абзацам, имеющим один уровень, им задаётся одинаковый формат.

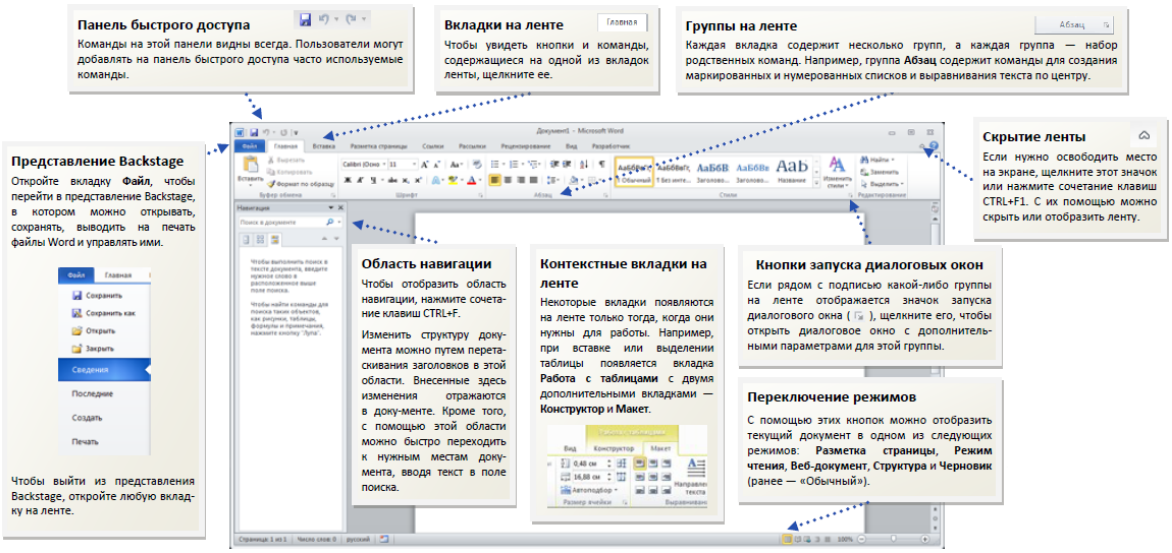

Рис. 1.1. Приложение Microsoft Office Word 2010

Стили могут не только задаваться фрагментам текста, но так же можно изменять параметры имеющихся стилей и создавать новые пользовательские стили. Параметры готовых стилей можно изменять как для использования только в текущем документе, так и сохранив внесённые изменения для работы с другими текстовыми документами.

Перемещая курсор мышки по представленным стилям, можно увидеть, как будет выглядеть выделенный фрагмент текста после применения того или иного стиля.

Чтобы выбрать другой набор необходимо использовать **Изменить стиль → Набор стилей**. Названия стилей во всех наборах одинаковые, изменяются только их параметры. Каждый из представленных стилей пользователь может отредактировать по своему усмотрению, изменив его параметры (например, задав другой шрифт или цвет). Можно создать новый стиль с совершенно уникальными параметрами.

Если нажать на кнопку «Стили» появится соответствующая панель (рис. 1.3). Чтобы узнать параметры каждого стиля, достаточно подвести курсор к его названию в списке – появится всплывающая подсказка с описанием.

При наведении курсора на название стиля справа появляется стрелочка, открывающая вспомогательное меню (рис. 1.4).

С его помощью можно найти в тексте случаи применения данного стиля в документе, отменить их, а также вызвать диалоговое окно редактирования стиля.

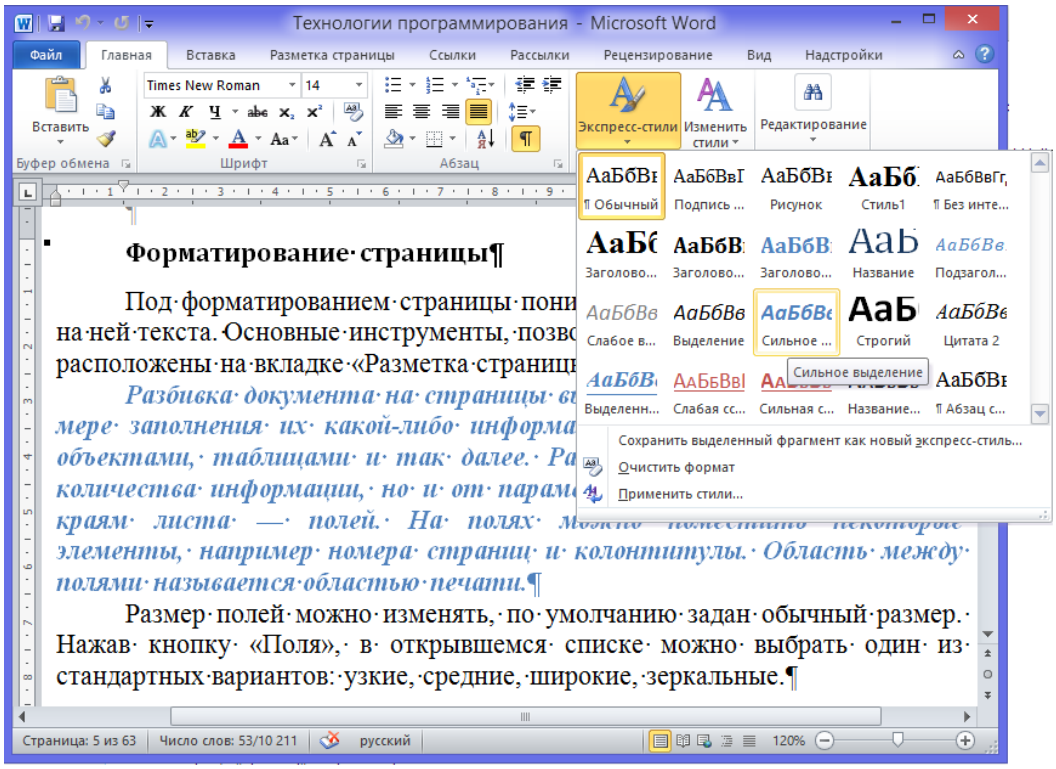

Рис. 1.2. Стили текста

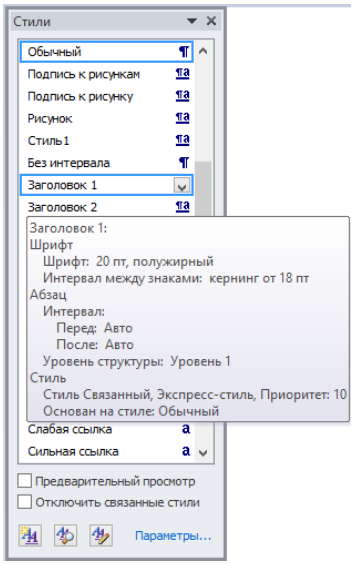

Рис. 1.3. Параметры стиля

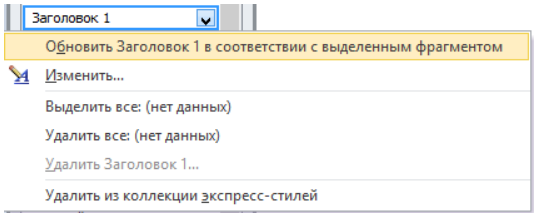

Рис. 1.4. Вспомогательное меню стиля

Чтобы создать новый стиль и задать ему необходимые параметры, необходимо воспользоваться кнопкой «Создать стиль»  . В открывшемся диалоговом окне нужно вести название нового стиля и выбрать фрагмент текста, к которому он будет применяться, например абзац, таблица, список. Если новый стиль создается на основе одного из базовых, то следует указать, какого именно. И наконец, нужно задать новому стилю оригинальные параметры: шрифт, его размер, цвет, начертание, тип выравнивания на странице, междустрочный интервал и так далее. В окне предварительного просмотра можно увидеть, как будет выглядеть стиль по мере изменения его параметров. Для сохранения созданного стиля в списке доступных, достаточно поставить флажок напротив надписи «Добавить в список экспресс-стилей». Чтобы отменить форматирование любого фрагмента текста, достаточно выделить его и нажать кнопку «Очистить формат» на панели «Шрифт».

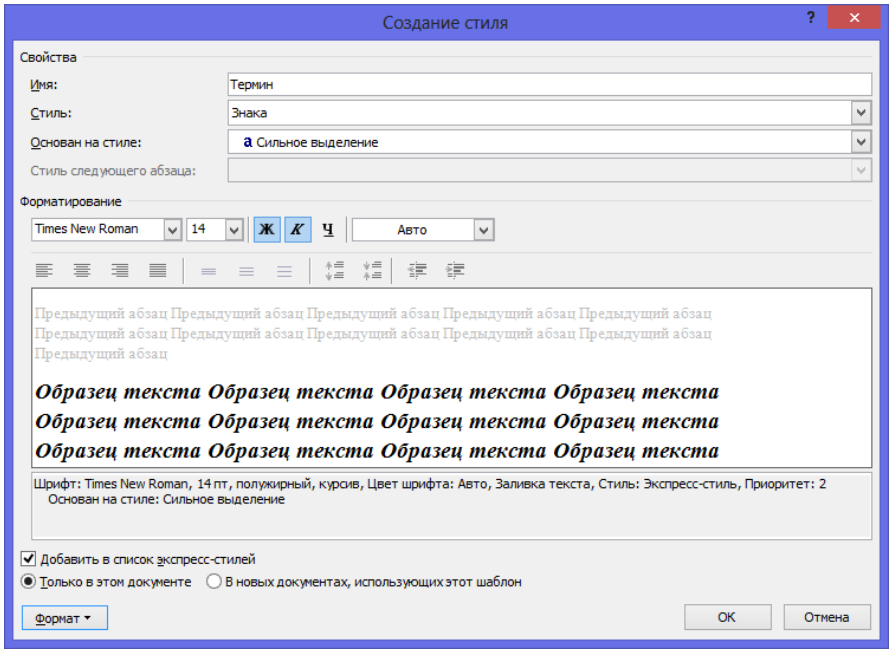

Рис. 1.5. Диалоговое окно «Создание стилей»

Стоит отметить, что при изменении параметров применённого стиля соответствующим образом изменятся все фрагменты текста, к которым данный стиль был применён.

Форматирование страницы

Под форматированием страницы понимается ее разметка и положение на ней текста. Основные инструменты, позволяющие задавать эти параметры, расположены на вкладке «Разметка страницы».

Разбивка документа на страницы выполняется автоматически, по мере заполнения их какой-либо информацией: текстом, графическими объектами, таблицами и так далее. Разбивка зависит не только от количества информации, но и от параметров пустых пространств по краям листа – полей. На полях можно поместить некоторые элементы, например номера страниц и колонтитулы. Область между полями называется областью печати.

Поля

Размер полей можно изменять, по умолчанию задан обычный размер. Нажав кнопку «Поля», в открывшемся списке можно выбрать один из стандартных вариантов: узкие, средние, широкие, зеркальные.

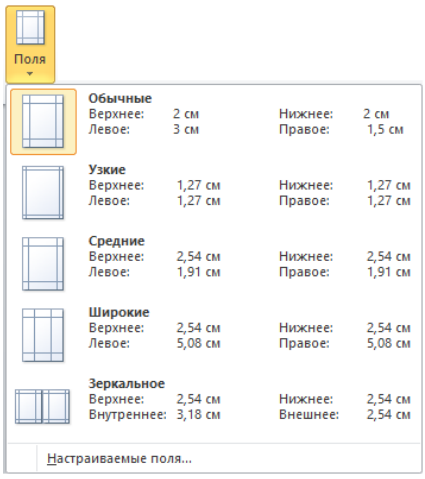

Рис. 1.6. Размеры полей документа

Если ни один из стандартных вариантов не подходит, можно выполнить более точную настройку параметров, выбрав «Настраиваемые поля».

Ориентация

Параметр «Ориентация» позволяет выбрать книжную ориентацию листов или же альбомную.

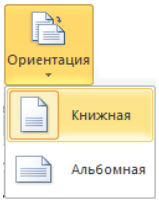

Рис. 1.7. Ориентация страницы

Размер

«Размер» позволяет задать размер листа, по умолчанию он соответствует А4. Если ни один из представленных размеров страницы не подходит, то можно задать пользовательский размер страницы («Другие размеры страниц»).

Колонки

Нередко возникает необходимость разбить текст на колонки. Это возможно сделать при помощи кнопки «Колонки» в группе инструментов «Параметры страницы».

Остальные кнопки на панели «Параметры страницы» позволяют задавать разрывы абзацев и страниц, нумерацию строк и включать/отключать функцию междустрочного переноса слов по слогам.

Фон страницы

Панель «Фон страницы» позволяет настраивать такие параметры, как подложка, цвет страницы и границы страниц.

1.3 Сноски и оглавления

Оглавления и сноски создаются при помощи инструментов на вкладке «Ссылки». Сноски бывают двух видов: подстраничные и концевые. Первые вставляются в нижней части той страницы, на которой встречается относимый к ним фрагмент текста, вторые выносятся в конец документа. Вставка сноски осуществляется при помощи кнопки «Вставить сноску». Для навигации между сносками используется меню .

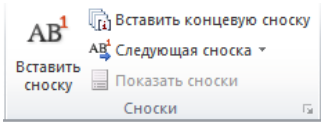

Рис. 1.7а. Вкладка «Ссылки»

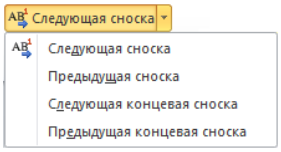

Рис. 1.7б. Меню «Следующая ссылка»

Нумерация сносок всегда сквозная и автоматически изменяется по мере их добавления или удаления. Чтобы прочитать текст сноски, необязательно прокручивать страницу вниз – достаточно подвести курсор к ее номеру в тексте, и появится всплывающее окошко.

Формат сносок можно изменить при помощи диалогового окна «Сноски» (рис. 1.8).

При работе с серьезными документами, например рукописями книг, дипломными проектами, деловыми отчетами, часто возникает необходимость создания оглавления. Это можно сделать автоматически при помощи панели «Оглавление» вкладки «Ссылки». Важно помнить, что для автоматического создания оглавления необходимо всем элементам его текста разного уровня следует задать одинаковый стиль. Например, имени автора присвоить стиль «Заголовок 1», названию книги – стиль «Заголовок 2», номерам или названиям глав – стиль «Заголовок 3» и так далее.

После присвоения всем элементам текста определённого стиля, нужно, щёлкнув по кнопке «Оглавление», выбрать один из вариантов оглавления (рис. 1.9).

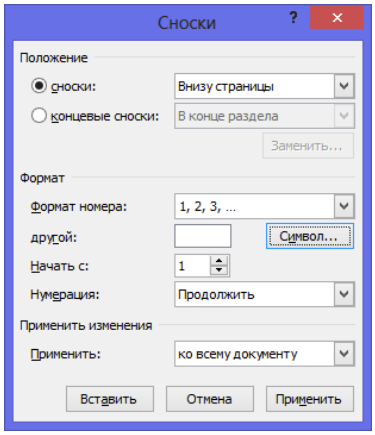

Рис. 1.8. Диалоговое окно «Сноски»

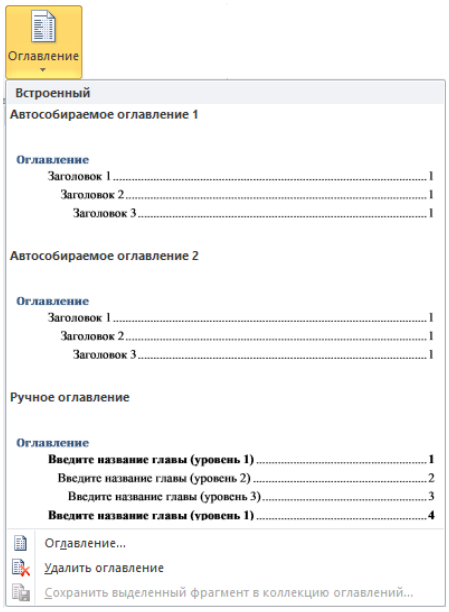

Рис. 1.9. Виды автоматически создаваемых оглавлений

Стоит отметить, что оглавление добавляется туда, где в тот момент находится текстовый курсор. А все вносимые в документ изменения будут отображаться в нём только после применения «Обновить таблицу» (вкладка «Ссылки», панель «Оглавление»).

Чтобы перейти к какому-либо разделу документа, указанному в оглавлении, нет необходимости прокручивать страницы вручную. Нужно всего лишь подвести курсор к его названию, зажать клавишу *Ctrl* и щелкнуть левой кнопкой мыши – программа автоматически переместит вас на нужную страницу.

Перекрёстные ссылки создаются при помощи инструмента «Перекрёстная ссылка» на панели «Названия» вкладки «Ссылки» или же инструмента «Перекрёстная ссылка» на панели «Связи» вкладки «Вставка».

1.4 Создание разделов

Раздел представляет собой часть документа, имеющую заданные параметры форматирования страницы. Новый раздел создается при необходимости изменения таких параметров, как нумерация строк, число столбцов или колонтитулы.

Для создания раздела необходимо щелчком мыши выбрать место, куда следует вставить разрыв раздела. Далее на вкладке «Разметка страницы» на панели «Параметры страницы» нужно выбрать инструмент «Разрывы», а в группе «Разрывы разделов» выбрать параметр, указывающий, откуда следует начать новый раздел. По мимо деления на разделы, при помощи данного инструмента возможно создавать разрывы страниц (рис. 1.10).

Для различных разделов одного текстового документа различные виды форматирования, как самого текста разделов, так и страниц документа, на которых данный раздел расположен. Так при работе с объемными таблицами и рисунками, выходящими за поля страницы, можно менять ориентацию страницы с книжной на альбомную.

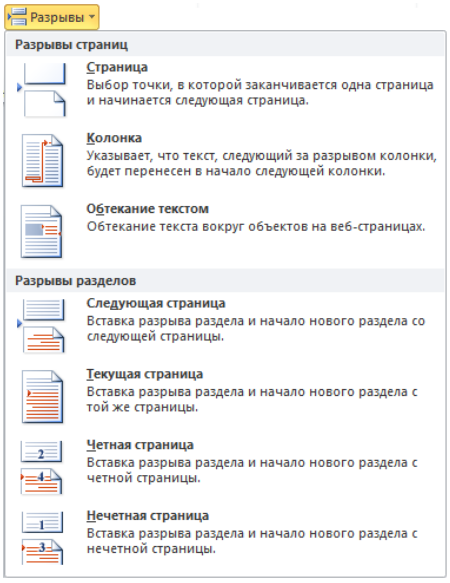

Рис. 1.10. Разрывы страниц/разделов

1.5 Колонтитулы

Колонтитулами называют какие-либо данные, помещенные вне основного текста на каждой странице и отображающиеся при распечатке. Чаще всего в этом качестве выступают заголовок книги, статьи или параграфа, фамилия автора, название фирмы и так далее. Колонтитулом может служить не только текст, но и изображение, например логотип компании. Если колонтитул расположен над текстом, его называют верхним, если под текстом – нижним.

Рис. 1.11. Колонтитул

Добавить колонтитул можно при помощи одноименной панели вкладки «Вставка» (рис. 1.12).

После вставки колонтитула активируется контекстно-зависимая вкладка «Работа с колонтитулами», на которой представлены основные инструменты для редактирования и форматирования колонтитулов (рис. 1.13).

Форматирование колонтитула осуществляется так же, как форматирование обычного текста. После двойного щелчка по колонтитулу, он становится доступным для редактирования. Перейти к режиму ввода и редактирования можно двойным щелчком по основному тексту. Чтобы задать разные колонтитулы для чётных и нечётных страниц, а так же для первой страницы необходимо задать соответствующие параметры на вкладке «Работа с колонтитулами» (панель «Параметры») (рис. 1.14).

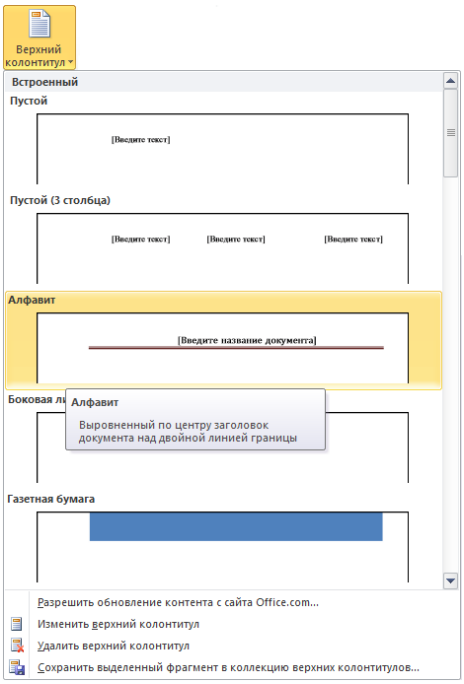

Рис. 1.12. Вставка колонтитула

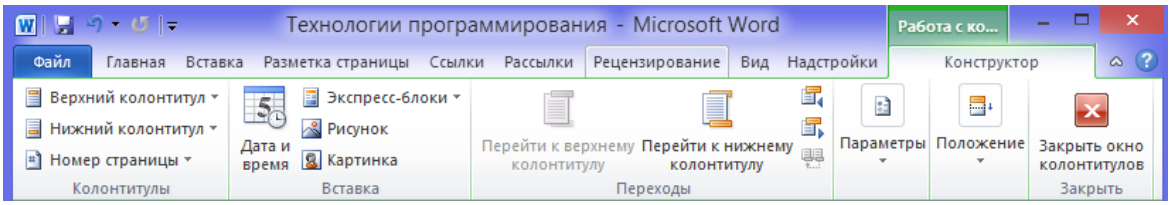

Рис. 1.13. Вкладка «Работа с колонтитулами»

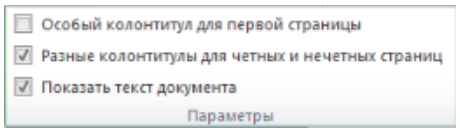

Рис. 1.14. Параметры колонтитулов

1.6 Нумерация страниц

Нумерация страниц в документе возможно несколькими способами: нумерация с первой страницы, нумерация не с первой страницы, нумерация внутри разделов. Номер страницы помещается в текст колонтитула, верхнего или нижнего в зависимости от выбранного расположения.

Задать нумерацию страниц можно используя инструмент «Номер страницы» на вкладке «Вставка» (рис. 1.15).

Формат номера страницы задаётся при помощи диалогового окна «Формат номера страницы».

Для нумерации внутри разделов необходимо поделить текст на разделы, если этого не было сделано ранее, выделить имеющиеся разделы, выбрать на вкладке «Вставка» инструмент «Номер страницы». В окне «Формат номера страницы» нужно поставить флажок «Включить номер главы» и в разделе «Нумерация страниц»

задать значение поля «начать с» равным первому номеру страницы, например 2 (рис. 1.16).

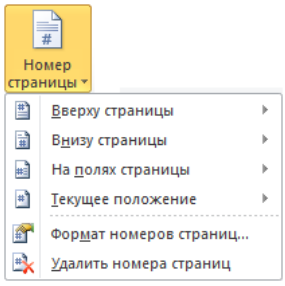

Рис. 1.15. Нумерация страниц

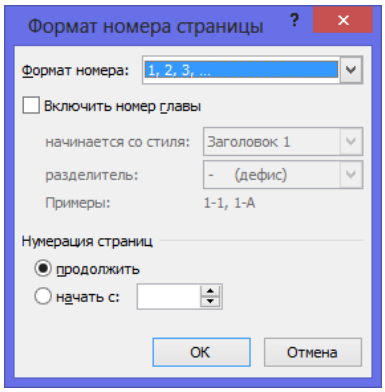

Рис. 1.16. Диалоговое окно «Формат номера страницы»

Для нумерации с учётом глав, необходимо в окне «Формат номера страницы» поставить флажок «Включить номер главы» определить стиль, а так же разделитель номера. В разделе «Нумерация страниц» задать нужное значение поля «начать с».

Для того чтобы убрать номер на первой странице и начать нумерацию со второй нужно, после того как страницы были пронумерованы, двойным щелком по колонтитулу, где расположен номер страницы, сделать его активным. Далее для него необходимо установить флажок «Особый колонтитул для первой страницы» на контекстной вкладке «Работа с колонтитулами» (панель «Параметры»).

Дополнительная теоретическая часть

1.7 Форматирование текста на уровне символов

Форматирование текста в Word 2010 осуществляется тремя способами: форматирование отдельных символов, форматирование абзацев и использование готовых экспресс-стилей. Первые два способа подразумевают изменение параметров выделенного текста по желанию пользователя – выбор шрифта, его размера, написания и цвета, типа выравнивания абзаца на странице и так далее. Использование одного из экспресс-стилей позволяет присвоить тексту уже готовый набор параметров, изменив также его структуру – обычный текст может превратиться в заголовок, подзаголовок, цитату и так далее.

Инструменты и команды для форматирования на уровне символов расположены на панели «Шрифт» вкладки «Главная».

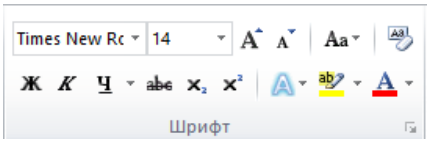

Рис. 1.17. Панель «Шрифт»

1.8 Форматирование текста на уровне абзацев

Опции и команды для форматирования абзацев расположены на панели «Абзац» вкладки «Главная». Они включают в себя функции выравнивания текста на странице, установку междустрочного интервала, абзацных отступов, маркированных, нумерованных и многоуровневых списков, границ текста, его сортировку и заливку фона абзаца. Для изменения формата нескольких абзацев их необходимо предварительно выделить. Без выделения новые параметры будут применены только к тому абзацу, в котором находится текстовый курсор.

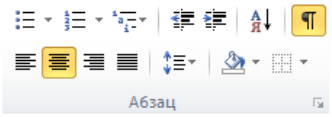

Рис. 1.18. Вкладка «Главная», панель «Абзац»

По умолчанию текст выравнивается по левому краю страницы без отступа красной строки. Отступ абзаца от левого края страницы можно изменять пошагово при помощи кнопки «Увеличить отступ». Для возвращения текста в предыдущее положение существует кнопка «Уменьшить отступ».

Междустрочным интервалом называют расстояние между строками. Интервал в 1,0 означает, что это расстояние равно высоте знаков в строке. По умолчанию для основного текста принят интервал в 1,15, для документов чаще всего используют интервал в 1,5. Выбрать другой показатель можно в меню кнопки «Интервал». Для более точной настройки параметров абзаца можно открыть диалоговое окно «Абзац» (рис. 1.19).

На вкладке «Отступы и интервалы» можно выбрать такие параметры, как выравнивание и уровень текста, отступы по правому и левому краю страницы и отступ красной строки, а также значение междустрочного интервала и интервала между абзацами.

На вкладке «Положение на странице» можно настроить такие параметры, как запрет висячих строк и разрыва абзаца, и указать исключения для форматирования, например запретить нумерацию строк и автоматический перенос слов на другую строку или страницу.

Текст, разделенный на абзацы, можно преобразовать в списки различного типа. Для этого на панели «Абзац» существуют три кнопки: маркеры , нумерация и многоуровневый список . Чтобы оформить несколько абзацев как список, их следует предварительно выделить. Если выделение отсутствует, то в пункт списка превратится только тот абзац, в котором стоит текстовый курсор.

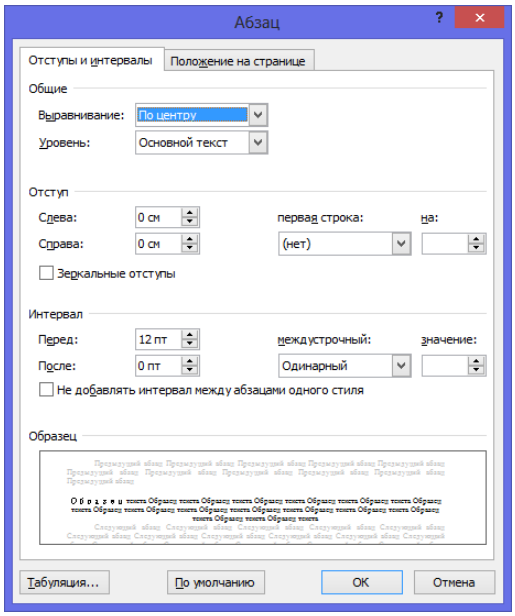

Рис. 1.19. Диалоговое окно «Абзац»

Чтобы добавить новый пункт в существующий список, достаточно поставить курсор в конец предыдущего абзаца и нажать клавишу Enter. Чтобы снова вернуть пункту списка вид абзаца, его следует выделить или установить в него текстовый курсор и еще раз нажать кнопку списка, возвращая ее в неактивное состояние.

При необходимости визуально вычленить один абзац из основного текста пользуйтесь кнопками заливки фона и границ текста. Для этого необходимо выделить нужный абзац, открыть меню кнопки «Заливка» и выбрать подходящий цвет из представленной палитры, затем открыть меню кнопки «Границы» и выбрать тип нужных границ, например «Внешние границы» (рис. 1.20).

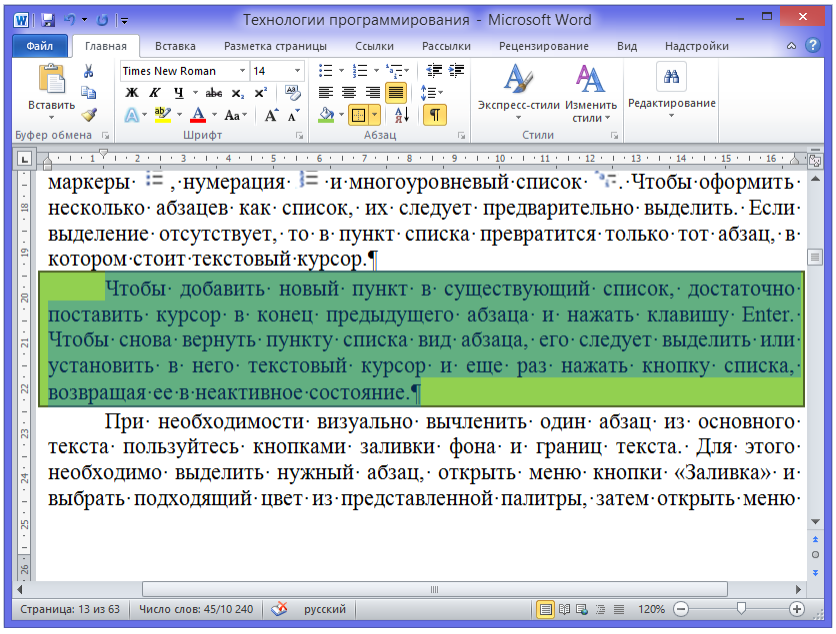

Рис. 1.20. Визуально выделенный абзац

1.9 Форматирование по образцу

Форматирование на уровне символов и абзацев создает массу трудностей, если приходится работать с документами большого объема. Изменить вручную параметры текста, состоящего всего из нескольких абзацев, не проблема. В случае если документ достаточно объёмный удобнее использовать преимущества форматирования по образцу. Кнопка вызова этой функции расположена на панели «Буфер обмена» вкладки «Главная». При этом необходимо форматировать нужным образом фрагмент текста, который и будет взят за образец.

1.10 Форматирование страницы. Подложка

Подложка – это надпись или изображения, помещаемые под текстом. Она может применяться для украшения документа или маркировки его состояния. В меню кнопки «Подложка» представлены стандартные варианты. Помимо этого, есть возможность создать собственные подложки. В меню кнопки «Подложка» представлены стандартные варианты. Помимо этого, есть возможность создать собственные подложки. Чтобы сделать это, достаточно выбрать «Настраиваемая подложка».

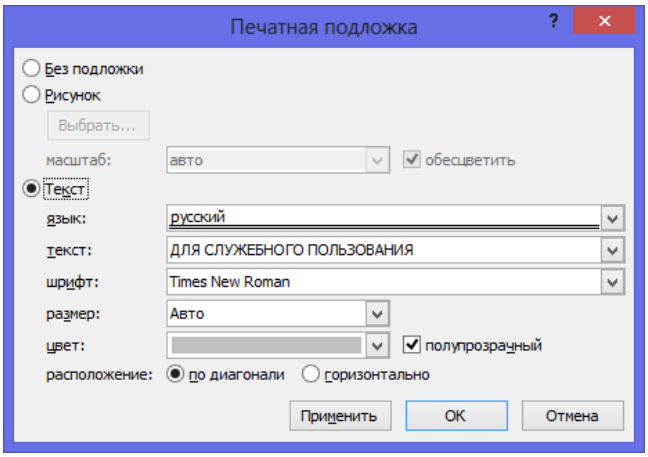

Рис. 1.21. Диалоговое окно «Печатная подложка»

Флажок напротив надписи «Текст» активирует текстовый режим подложки. В окне можно выбирать шрифт, его размер и цвет, изменять язык и саму надпись. Вместо подложки можно использовать фоновую заливку страницы. В опциях кнопки «Цвет страницы» доступна широкая палитра цветов и оттенков. Чтобы увидеть предварительный результат, достаточно навести курсор мыши на один из цветов. Обычный выбор цвета дает монотонную заливку. В некоторых случаях гораздо эффектнее смотрятся градиентная, узорная и текстурная заливки. Чтобы настроить их параметры необходимо использовать «Способы заливки».

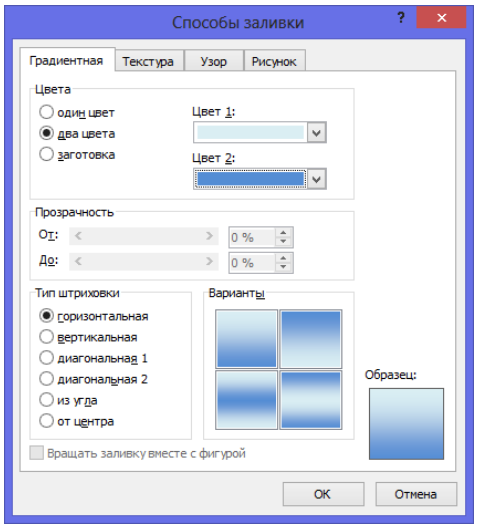

Рис. 1.22. Диалоговое окно «Способы заливки»

При необходимости можно использовать рамки «Границы страницы».

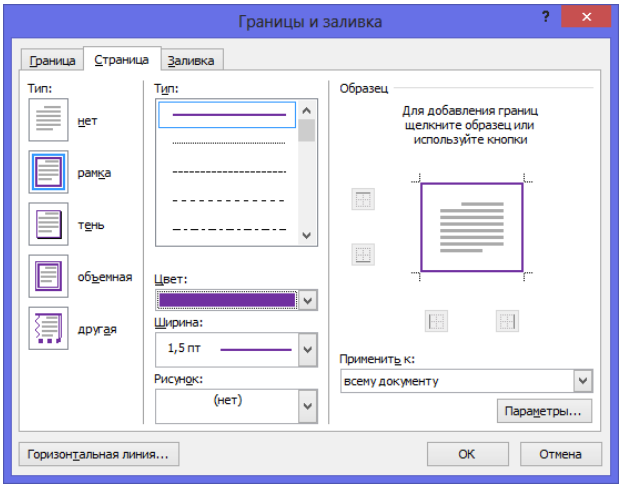

Рис. 1.23. Диалоговое окно «Границы и заливка»

1.11 Вставка символов и формул

Не все символы можно ввести с клавиатуры. Специфические символы и значки доступны только в меню вкладки «Вставка» (рис. 1.24).

По умолчанию в выпадающем списке отображается 20 значков, которые были использованы последними. Если среди них нет нужного, его можно поискать в меню кнопки «Другие символы».

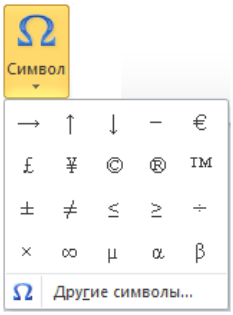

Рис. 1.24. Меню «Символ»

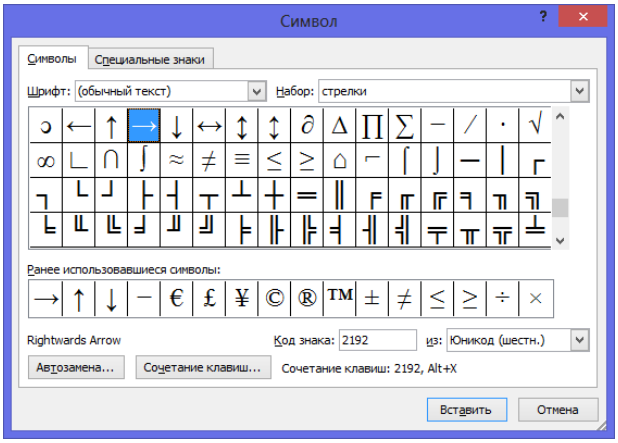

Рис. 1.25. Диалоговое окно «Символ»

Вставку многих символов можно осуществлять при помощи функции автозамены, что ощутимо ускоряет набор текста. Для этого в меню кнопки «Другие символы» достаточно нажать кнопку «Автозамена». В правой части открывшегося окна указаны символы, которые нужно ввести, а в левой – комбинации клавиш, посредством которых вводится каждый символ.

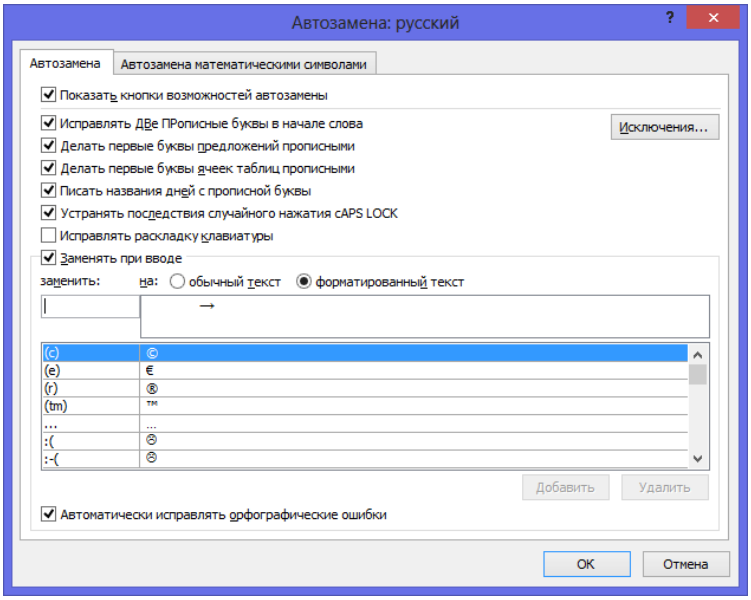

Рис. 1.26. Диалоговое окно «Автозамена»

В меню кнопки «Формула», которое открывается щелчком по ее нижней части, можно найти несколько готовых формул. Чтобы создать новую формулу, нужно щёлкнуть по верхней части кнопки «Формула». В документ будет вставлено поле для ввода формулы, а на ленте управления появится вспомогательная вкладка «Работа с формулами», состоящая из инструментов для редактирования.

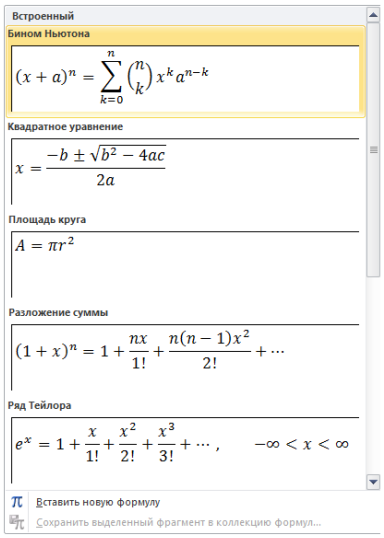

Рис. 1.27. Создание формул

Задание

1. Открыть файл «Тенденции на рынке труда.doc» в MS Word 2010.
2. В данном документе необходимо сделать:

а) Выравнивание по ширине;

б) Название статьи, приведённой в документе, вынести на первую, текст самой статьи (с подзаголовка «Экономическая активность населения») должен начинаться со второй страницы. В данном случае для этого можно выделить название статьи в отдельный раздел, применив разрыв «Следующая страница».

в) Присвоить элементам текста разного уровня соответствующие стили. Основной заголовок – «Заголовок 1», подзаголовок – «Заголовок 2», основной текст – «Обычный» и т.д.

г) После присвоения стилей, используя панель «Стили» вкладки «Главная» изменить существующие параметры, применённых стилей.

- Для стиля «Заголовок 1»: Шрифт – Arial Black, начертание – курсив, размер шрифта – 20, цвет – оттенок зелёного.
- Для стиля «Заголовок 2»: Шрифт – Calibri, начертание – полужирный курсив, размер шрифта – 16, цвет – оттенок синего.
- Для стиля «Обычный»: Шрифт – Times New Roman, начертание – обычный, размер шрифта – 14, цвет – чёрный, отступ перед первой строкой абзаца 1.3, междустрочный интервал 1.5 (отступ и интервал задаются при помощи «Формат» → «Абзац»).

д) Для текста, расположенного в таблице, создать новый стиль. Необходимо задать параметры:

- *имя* – имя стиля;

- *стиль* – элемент текста, для которого создаётся стиль: абзац, знак, таблица, список;
- *основан на стиле* – пользователь может выбрать стиль-образец;
- параметры шрифта.

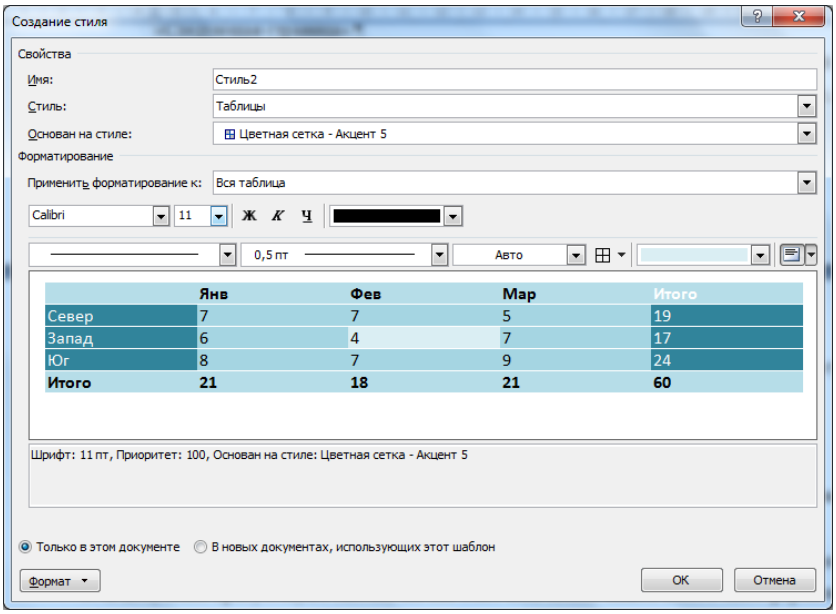

Рис. 1.28. Создание стиля

е) Полученный текст поделить на соответствующие разделы.

3. После применения стилей к элементам текста разного уровня, необходимо пронумеровать страницы, при этом номера на первой странице быть не должно. Для этого после применения инструмента «Номер страницы», расположенного на вкладке «Вставка», необходимо в появившейся контекстной вкладке «Работа с колонтитулами» на панели «Параметры» необходимо поставить флажок «Особый колонтитул для первой страницы».

4. Далее для редактируемого документа нужно создать содержание, поместив его на второй странице. Для этого необходимо воспользоваться инструментом «Оглавление» на вкладке «Ссылки». Если стили заголовков применены правильно, то в содержании будут отображены все имеющиеся разделы.

5. В документе присутствуют сноски, так как текст изначально лишён какого-либо форматирования, они выглядят как обычный текст. Поэтому для слов, к которым они относятся, нужно создать новые сноски с соответствующими значениями, и удалить из основного текста ранее созданные. Для создания сносок воспользоваться инструментом «Вставить сноску» на вкладке «Ссылки».

6. Имеющаяся таблица выходит за поля документа, поэтому необходимо для страницы, содержащей таблицу, изменить ориентацию с книжной на альбомную. Для этого таблицу нужно поместить в отдельный раздел, размещающийся на отдельной странице и изменить соответствующим образом ориентацию данной страницы.

Контрольные вопросы

1. Что такое стиль в документах MS Word?
2. Чем удобно использование стилей?
3. Чем отличаются разрыв страницы и разрыв раздела?
4. Чем удобно использование разрывов разделов?
5. Для чего в тексте используются колонтитулы?

Список литературы

1. Поддержка Office [Электронный ресурс]. – Режим доступа: https://support.office.com/ru-ru/home/, свободный.

2. Обучающие курсы, видео и учебники по Word 2013 [Электронный ресурс]. – Режим доступа: https://support.office.com/ru-ru/article/, свободный.

Лабораторная работа № 2
Системы контроля версий

Цель работы:

Получить опыт практической работы с системой контроля версий на примере BitBucket и TortoiseGit.

Введение

Контроль версий подразумевает под собой комплекс методов, направленных на систематизацию изменений, вносимых разработчиками в программный продукт в процессе его разработки и сопровождения, сохранение целостности системы после изменений, предотвращение нежелательных и непредсказуемых эффектов. Также использование систем контроля версий позволяет сделать процесс внесения изменений более формальным.

Система управления версиями гарантирует, что каждый автор всегда работает с самой последней версией файла, а также исключает возможность случайной перезаписи любым из авторов работы своих коллег.

BitBucket – это сервис для разработчиков программного обеспечения, предоставляющий хостинг для проектов.

Одним из основных сервисов, предоставляемых BitBucket, является контроль версий. Доступные системы контроля версий – Git и Mercurial. Также доступны сервисы просмотра исходного кода, issue-трекинга, ведения wiki проекта и другие.

2.1 Создание проекта на BitBucket

1. Зайдите на сайт https://bitbucket.org. Авторизируйтесь под своим аккауном google, либо создайте новый аккаунт. При регистрации необходимо выбрать «5 users» в пункте «Plan».
2. Для воздания частного репозитория необходимо нажать **Create → Create Repository** в верхнем правом углу на главной странице (рис. 2.1). Однако в данной лабораторной работе рекомендуется сначала создать команду через пункт меню **Teams → Create team**. На данном этапе вы можете не добавлять никого в команду (это понадобится для выполнения лаб. раб. №6). После создания команды вам будет предложено создать репозиторий.

В открывшейся странице (рис. 2.2):

- введите название проекта в поле «Name». Для того чтобы осмысленно выбрать название проекта (что является важным) необходимо определится с вариантом задания (см. лаб. раб №3)
- введите краткое описание в поле «Description»
- выберите тип репозитория «Git» в поле «Repository type»
- отметьте пункт «Issue tracking» в поле «Project Management»
- выберите язык программирования «C#» в поле «Language»
- нажмите кнопку «Create Repository»

После создания репозитория будет доступно его меню (рис. 2.1, слева). Меню состоит из двух групп «Actions» и «Navigation». Пункт меню «Clone» в группе «Ac-

tions» позволяет получить адрес репозитория для дальнейшей работы с ним. Пункт меню «Source» в группе «Actions» предназначен для просмотра исходного кода, пункт меню «Commits» позволяет просмотреть историю изменений, пункт «Issues» предназначен для учета ошибок и задач. Кроме того есть возможно скачать содержимое репозитория с помощью пункта «Downloads».

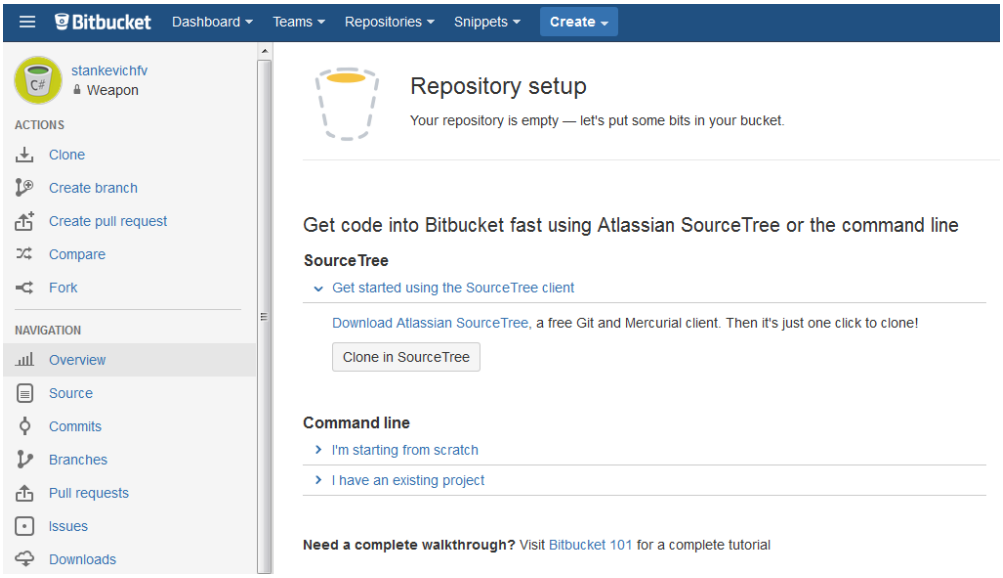

Рис. 2.1 Главная страница BitBucket

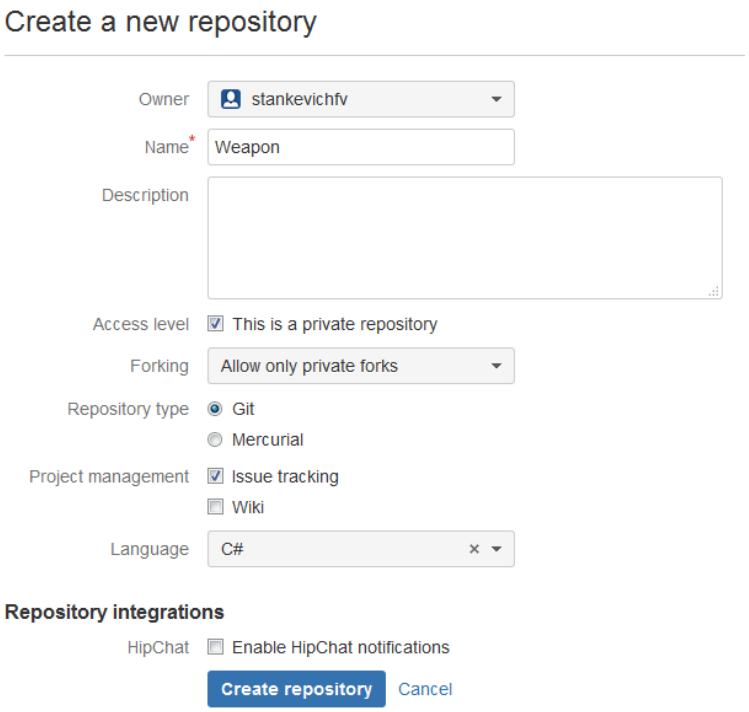

Рис. 2.2. Создание репозитория

2.2 Система контроля версий Git

BitBucket – это централизированная система управления версиями. В настоящее время она используется многими сообществами разработчиков программного обеспечения. BitBucket предоставляет возможность создания как открытых, так и закрытых репозиториев. Создание закрытых репозиторией, рассчитанных на число пользователей более пяти, являются платной услугой.

Git является распределённой системой управления версиями. Распределённость подразумевает хранение исходного кода в разных хранилищах. Хранилища могут располагаться как на локальном диске, так и на сетевом сервере.

Для организации совместной работы над файлами в большинстве системах контроля версий используется модель копирование – изменение – слияние. Для файлов, не допускающих слияние (например, бинарные файлы), можно использовать модель блокирование – изменение – разблокирование. Однако система Git ориентирована на работу не с самим файлами как таковыми, а с их изменениями, то есть «единицей обработки» для Git является «изменение файла».

Для работы с Git будем использовать бесплатный клиент TortoiseGit, выполненный как расширение оболочки Windows.

2.3 Начало работы с BitBucket

Для того чтобы скачать репозиторий проекта, необходимо сделать следующее:

1. Создайте папку, в которой будет храниться ваша локальная копия проекта (для удобства лучше хранить все репозитории в одной папке, например, C:\Git).
2. Нажимаем на созданную папку правой кнопкой мыши и из контекстного меню выбираем «Git Clone…» (рис. 2.4).

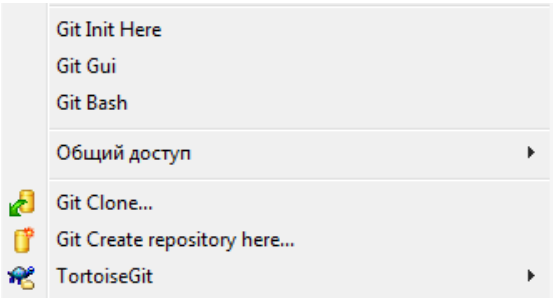

Рис. 2.4. Контекстное меню папки

3. В появившемся диалоговом окне (рис. 2.5) в поле «URL» введите полный URL репозитория Git (https://..., без команды *git clone*). (рис. 2.1, пункт «Clone»)
4. Нажмите кнопку «OK», начнется скачивание репозитория с Git-сервера.
5. Затем система попросить ввести пароль от репозитория BitBucket.

Модификация проекта

С локальной рабочей копией можно работать как с обычной папкой: создавать, редактировать, удалять файлы и/или папки.

1. Добавим новый файл в папку проекта (файл SampleProgram.cs).

2. Нажмите правой кнопкой мыши по папке проекта, из контекстного меню выберите «Git Commit - > “master”…» (рис 2.6).

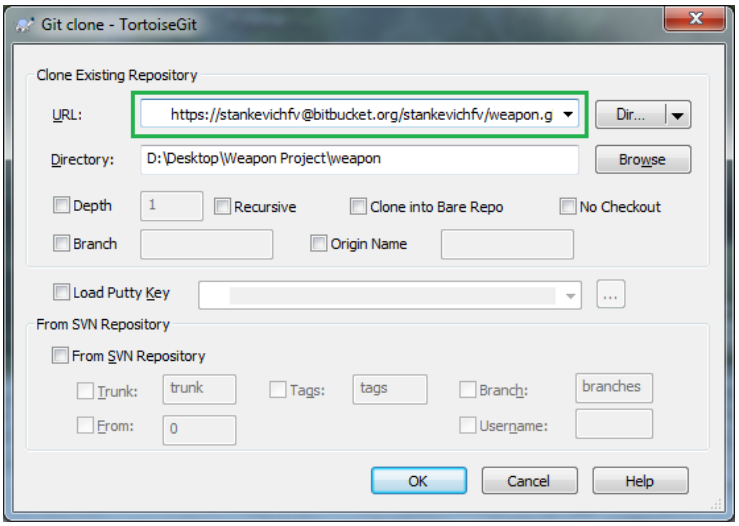

Рис. 2.5. Диалоговое окно «Git clone»

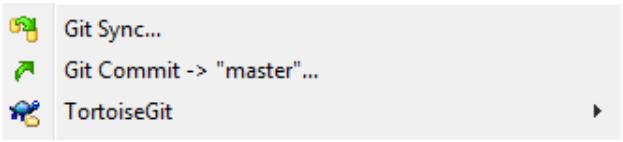

Рис. 2.6. Контекстное меню

3. При первом коммите TortoiseGit запросит ввести имя пользователя и электронную почту (рис. 2.7)

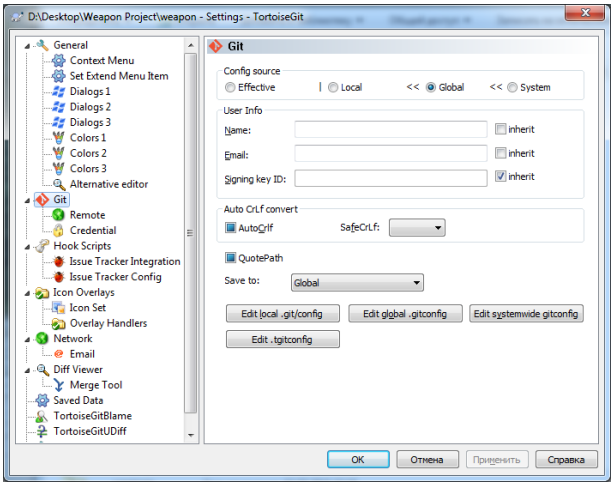

Рис. 2.7. Ввод имени и e-mail

4. В открывшемся окне (рис. 2.8) в секции «Message» введите краткое описание внесенных изменений (это рекомендуется делать всегда, особенно если не вы один работаете над проектом).
5. В секции «Changes made (double-click on file for diff)» вы увидите список всех внесенных изменённых: добавленных, измененных и удалённых фай-

лов локальной рабочей копии. У изменённых файлов будут автоматически отмечены флажком, остальные – нет. Установите флажки у тех файлов, изменения которых должны быть загружены на git-сервер (есть флажок – файл обновляется, добавляется, удаляется; нет флажка – остаётся без изменений). Двойной щелчок по файлу запустит утилиту Tortoise Merge, предназначенную для сравнения версий файлов, которая покажет последнюю версию файла из репозитория Git в левом окне и текущую рабочую копию в правом. Изменения будут выделены: удалённые строки зачёркнуты, добавленные выделены.

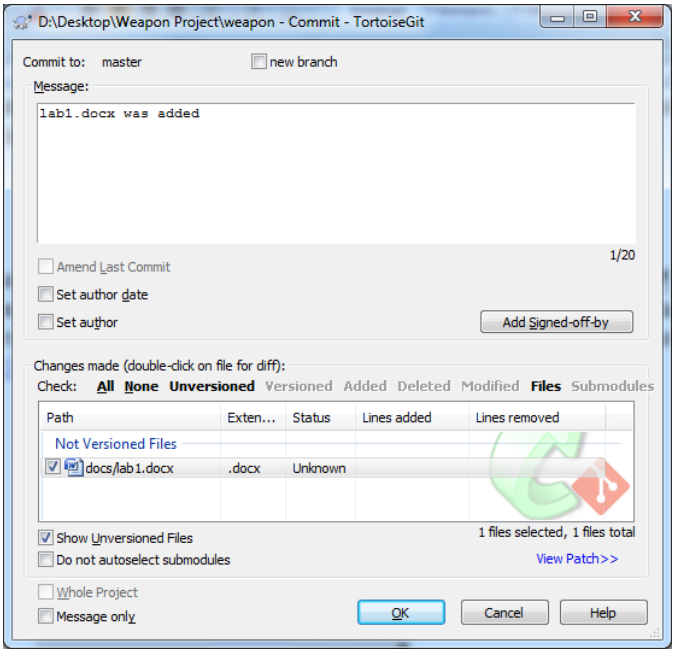

Рис. 2.8. Диалоговое окно «Commit».

6. После выделения нужных файлов или всех сразу, нажмите кнопку «OK». Произойдет фиксация внесенных вами изменений в локальном репозитории. Каждый раз при выполнении фиксации в репозитории создаётся новое состояние дерева файловой системы, называемое *ревизией*. Каждой ревизии назначается уникальный номер. Начальная ревизия не содержит ничего, кроме пустой корневой папки.
7. Для того чтобы отправить изменения на сервер необходимо нажать правой кнопкой мыши по папке проекта снова, из контекстного меню выбрать «Git Sync» (рис. 2.6).
8. В появившемся окне (рис. 2.9) нажмите «Push» и введите пароль.

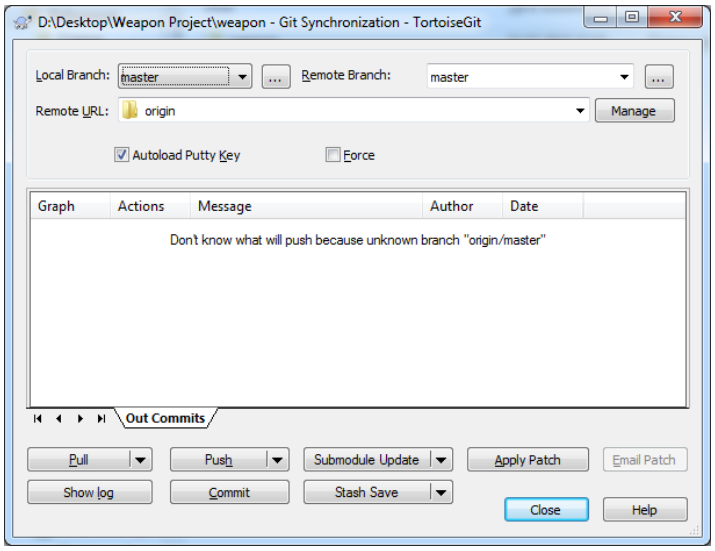

Рис. 2.9. Диалоговое окно «Git Sync».

Просмотр изменений

Откройте добавленный ранее файл (SampleProgram.cs) в любом текстовом редакторе и внесите в него изменение. Выполните «Commit» и «Push», как было описано выше.

Для просмотра изменений в проекте можно использовать инструменты сайта https://bitbucket.org (пункт меню «Commits» рис. 2.10 изменения выделяются цветом), так и возможности TortoiseGit (рис. 2.11 – 2.12).

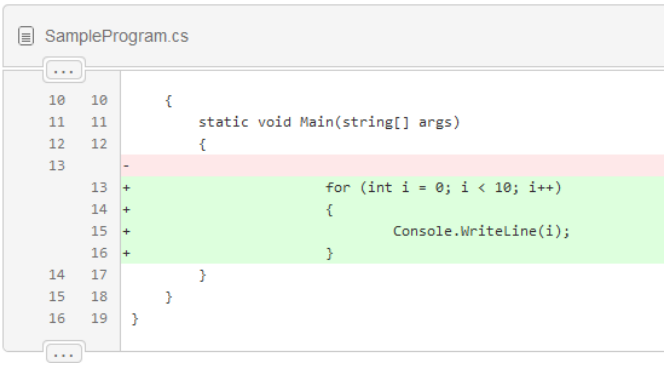

Рис. 2.10. Просмотр изменений

Для просмотра изменений средствами TortoiseGit необходимо вызвать контекстное меню любой папки или файла проекта (рис. 2.11).

TortoiseGit позволяет просмотреть журнал изменений любого файла/каталога (пункт меню «Show log») или построить граф ревизий (пункт меню «Revision graph»). Граф ревизий полезен для больших проектов с несколькими отдельными ветвями разработки. Для отображения изменений в программном коде используется журнал изменений (рис. 2.12). Он показывает список всех фиксаций изменений в файле или папке, а также детальные сообщения разработчиков.

После выбора «Show log» для просмотра отдельных изменений в коде щелкните два раза на изменении из списка. В результате откроется окно утилиты Tortoise Merge, отображающее внесенные изменения (рис. 2.13).

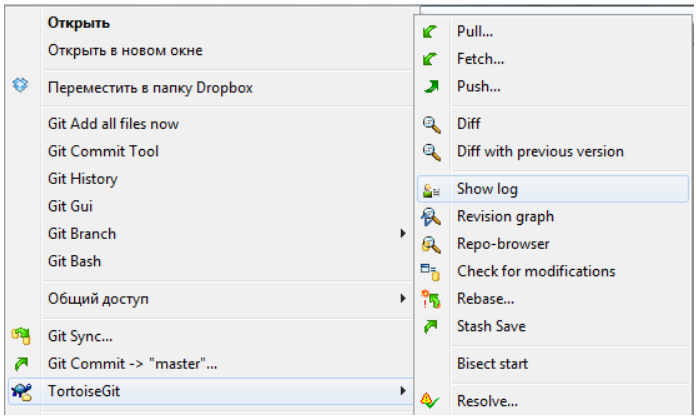

Рис. 2.11. Контекстное меню

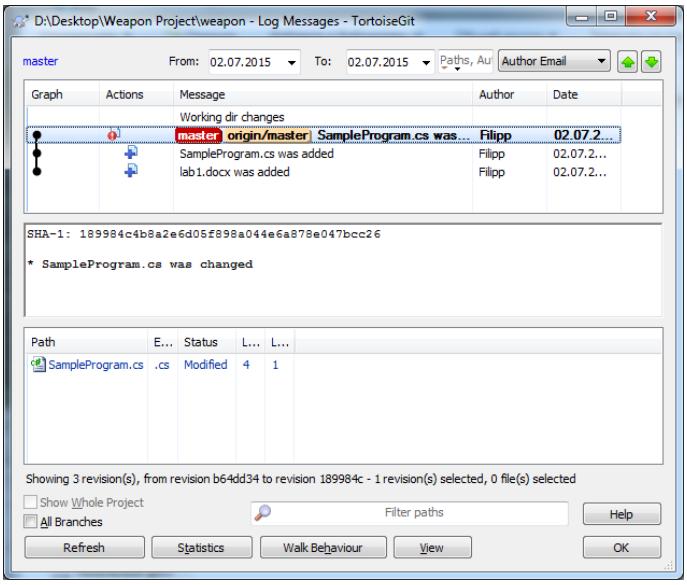

Рис. 2.12. Журнал изменений

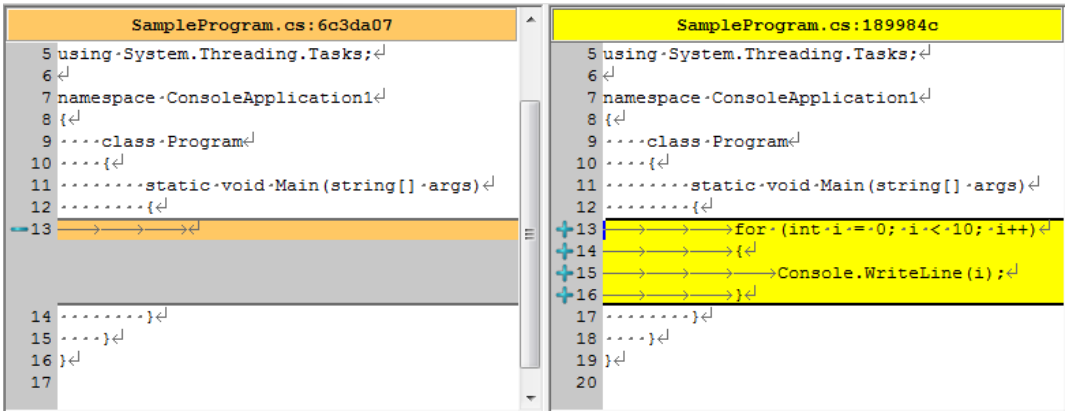

Рис. 2.13. Внесенные изменения

Получение последней версии проекта

Для получения изменений с сервера необходимо нажать кнопку «Pull» в диалоговом окне «Git Sync» (рис. 2.9) и дождаться, пока изменения загрузятся с сервера. Рекомендуется всегда перед «Commit» выполнять действие «Pull». Также «Pull» можно выполнить посредством контекстного меню папки проекта пункт **TortoiseGit → Pull** (рис. 2.11).

Структура проекта в хранилище

При разработке программного обеспечения основная задача делится на более мелкие, которые выполняются либо последовательно, либо параллельно относительно друг друга. При разработке выбирается основная ветвь разработки (в нее входят наиболее важные этапы, оказывающие влияние на весь проект), от нее отходят дочерние ветви, содержащие менее важные процессы. Таким образом, при разработке программных продуктов необходимо обеспечивать возможность одновременной разработки. Стратегически грамотное ветвление позволяет сохранять порядок и последовательность при работе с большим числом версий программного обеспечения.

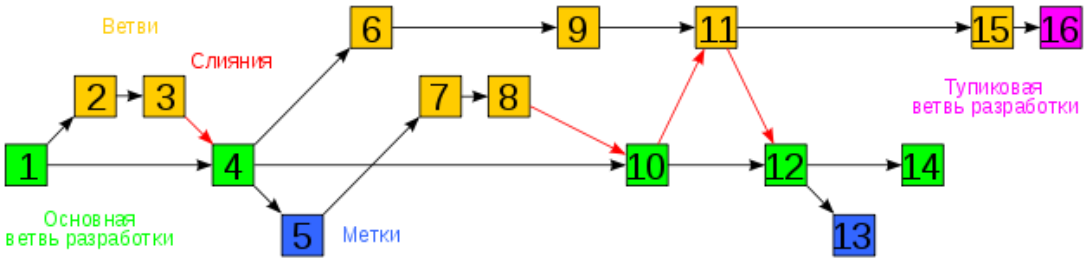

Рис. 2.14. Пример эволюции ветвей

Формально Git не накладывает каких-либо ограничений на файловую структуру проекта – она может быть какой угодно в рамках правил именования объектов файловой системы. Тем не менее, существуют рекомендации, призванные облегчить работу с ветвями и метками.

Основанная линия разработки проекта называется *trunk*. Дочерние же линии проекта именуются *branches*. Имя конкретной ветки (*branch*) можно выбрать в соответствии с разрабатываемым в ней функционалом, например *branch_new_gui*.

Задание

1. Создайте новый проект на BitBucket.
2. Используя TortoiseGit, поместите в него текстовый документ, созданный в предыдущей лабораторной работе.
3. Работа в паре – первый участник вносит изменение в документ – второй просматривает изменения, вносит исправления. Просмотр истории изменений.

Требования и рекомендации:

Название проекта и все комментарии к коммитам должны быть осмысленными. Это необходимо для того чтобы было удобнее понимать суть проекта и изменений, а также отслеживать их. Кроме того это поможет человеку, увидевшему проект в первый раз, разобраться в нем.

Контрольные вопросы

1. Что такое системы контроля версий?
2. Какие системы контроля версий вы знаете?
3. Какие существуют основные операции в системе контроля версий?

Список литературы

1. Scott Chacon. Pro Git [Электронный ресурс]. – Режим доступа: https://git-scm.com/book/ru/v2, свободный.

2. Mike McQuaid. Git in Practice, Manning, 2014.

3. Bitbucket Documentation [Электронный ресурс]. – Режим доступа: https://confluence.atlassian.com/display/BITBUCKET/Bitbucket+Documentation+Home, свободный.

Лабораторная работа № 3
Создание объектной модели предметной области

Цель работы:

Получить опыт практической работы в создании иерархий классов предметной области с помощью UML диаграмм.

Введение

UML (Unified Modeling Language – унифицированный язык моделирования) представляет собой язык объектного моделирования, в котором используются графические элементы для визуального проектирования и создания абстрактной модели разрабатываемой системы.

Язык UML применяется во многих областях, таких как описание бизнес-процессов, системное проектирование, визуализация организационных структур, разработка программного обеспечения и т.д. Данный язык не является языком программирования, но на основе реализованных UML-диаграмм возможно сгенерировать программный код.

3.1 Сущности в UML

Для проектирования абстрактных моделей в языке UML используются сущности.

Объектом называется именованная сущность, которая обладает свойствами и определенным образом проявляет свое поведение. Объекты являются экземплярами класса.

Класс содержит описание некоторого набора реально существующих объектов, которые между собой связаны по определенным критериям. Так же класс содержит описание некоторого набора осуществляемых над объектами операций.

На рисунке 3.1 и 3.2 в качестве примера изображен чертеж танка и его описание в виде графического элемента языка UML.

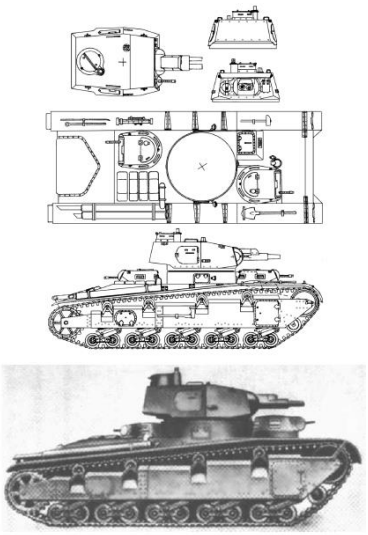

Рис.3.1. Описание класса (чертеж) и реальный объект

Принципы объектно-ориентированного программирования заключаются в следующих понятиях:

- абстрагирования данных;
- инкапсуляции;
- наследования;
- полиморфизма;
- «позднего связывания».

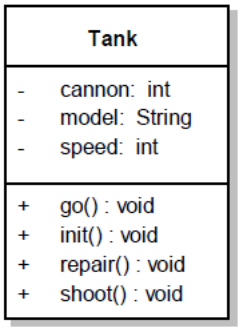

Рис.3.2. Графическое изображение класса

Инкапсуляцией (encapsulation) называется принцип, направленный на объединение данных и кода, с помощью которого осуществляется манипуляция этими данными, а также защита данных от неправильного использования и прямого внешнего доступа. Другими словами, манипуляции данными класса осуществляется с помощью методов этого же класса.

Наследованием (inheritance) является процесс, с помощью которого один объект может наследовать свойства другого объекта. Другими словами, объект может приобретать свойства, присущие другому объекту и добавлять к ним определенные свойства и методы, характерные только для него.

Наследование можно разделить на два вида:

- одиночное наследование – это класс, который получил наследование только от одного суперкласса (предка);
- множественное наследование – это класс, который может получить наследование от любого количества предков (в C# запрещено).

На рисунке 3.3 изображен класс «Auto», который является суперклассом, а класс «Tank» – подклассом.

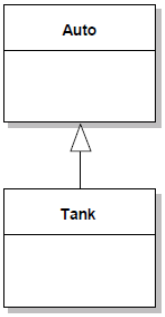

Рис. 3.3. Графическое изображение наследования

Полиморфизмом (polymorphism) называется механизм, который направлен на использование одного и того же имени метода для осуществления нескольких похожих, но отличающихся задач. Цель полиморфизма в объектно-ориентированном программировании заключается в использовании одного имени для задания общих действий, присущих классу. Другими словами концепцию полиморфизма можно выразить в виде идеи «один интерфейс, множество методов».

С помощью механизма «позднего связывания» определяется принадлежность объекта конкретному классу, а так же производится вызов метода, который имеет отношение к классу и объект которого был использован.

Таким образом, процессам наследования и полиморфизма присуща следующая парадигма: объект подкласса может использоваться везде, где используется объект суперкласса.

Если осуществляется вызов метода класса, то он ищется в самом классе. Если данный метод существует в текущем классе, то он вызывается. Иначе выполняется обращение к родительскому классу и в нём производится поиск вызываемого метода. Если в родительском классе поиск так же не увенчался успехом, то продолжается обращение вверх по иерархии классов до самого верхнего уровня [1].

Класс имеет название и содержит набор операций и атрибутов. На диаграмме UML класс отображается в виде прямоугольника, который разделен на три области. Верхняя область отображает название класса. В средней области отображается описание атрибутов. Нижняя область отображает описание операций, выполняемых объектами данного класса (рис. 3.4).

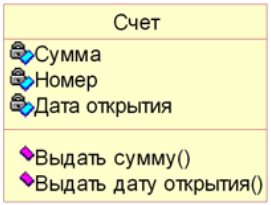

Рис. 3.4. Изображение класса в нотации UML

С помощью **атрибутов** класса отображается состав и структура данных, которые хранят объекты класса. Каждый атрибут представляется в виде имени и типа, определяющего данные, которые он содержит. При программной реализации объекта выделяется память, которая необходима для хранения всех атрибутов и таким образом, каждый атрибут имеет конкретное значение в любой момент времени выполнения данной программы. В программе может содержаться множество объектов, относящихся к определенному классу. Все эти объекты имеют одинаковый набор атрибутов, который описан в классе, однако значения атрибутов могут быть индивидуальными и изменяться в процессе выполнения программы. Атрибутам класса можно присвоить видимость (visibility). Данная характеристика отображает степень доступности каждого атрибута для других классов. В языке UML определено несколько степеней видимости для атрибутов:

- Открытый (public) – атрибут полностью открыт и доступен для других классов (объектов);
- Защищенный (protected) – атрибут открыт и доступен только для потомков данного класса;

- Закрытый (private) – атрибут является не доступным для внешних классов (объектов).

С помощью последнего значение возможна реализация свойства инкапсуляции. Например, в целях несанкционированного доступа можно сделать все атрибуты класса закрытыми, полностью скрыв данные класса от внешнего мира. В данном случае изменение атрибутов класса никак не скажется на остальной части программы, что позволяет сократить количество ошибок в программе.

Так же в классе содержатся объявления операций, представляющие собой запросы, которые выполняются объектами данного класса. Каждая операция содержит описание своих параметров: имя операции, возвращаемое значение и список параметров.

Так же как и для атрибутов класса, для операций определен режим видимости. Операции могут быть закрытыми и не доступными для внешних классов (объектов). Остальные операции являются интерфейсной частью класса и представляют собой средство интеграции в ПС. В идеале каждый класс направлен на выполнение чего-то одного. Необходимо избегать слишком больших, громоздких или слишком маленьких классов. В случае если абстрактные классы слишком большие, то это вносит определенные сложности в возможность модификации и изменения модели. Если же классы слишком маленькие, то это увеличивает количество абстракций и усложняет логическое понимание и управление ими.

3.2 Отношения между сущностями

Логические связи между сущностями в диаграммах классов задаются с помощью **отношений.** В языке UML существует несколько видов отношений.

Ассоциация

С помощью **ассоциации** определяется взаимосвязь объектов одной сущности (класса) с объектами другой сущности. Наиболее часто для связывания двух классов используются бинарные ассоциации. Ассоциация может иметь название, отражающее суть данной связи. Так же ассоциация может обладать такой характеристикой, как множественность. Данная характеристика показывает количество объектов каждого класса имеющих возможность участвовать в ассоциации. Множественность отображается на концах ассоциации и представляется в виде конкретного числа или числового диапазона. Множественность может быть задана в виде звездочки, что означает любое количество объектов класса.

Например, ассоциация связывает один объект класса «Набор товаров» с одним или более объектами класса «товар» (рис. 3.5).

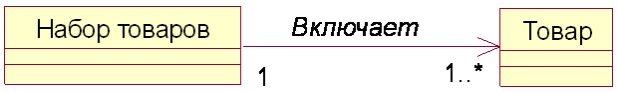

Рис.3.5. Применение ассоциаций

Связаны между собой могут быть и объекты одного класса, поэтому ассоциация может связывать класс с самим собой. Например, для класса «Житель города» можно ввести ассоциацию «Соседство», которая позволит находить всех соседей конкретного жителя.

В языке UML существует несколько типов ассоциации. Самыми распространенными типами являются однонаправленная и двунаправленная. Например, двунаправленная ассоциация связывает классы «рейс» и «самолёт». Однонаправленная ассоциация связывает классы «человек» и «кофейный автомат».

Двунаправленные ассоциации отображаются в виде линии с двумя концами, соединяющей два классовых блока. Ассоциации, обладающие более высокой степенью направленности, имеют более двух концов и представляются в виде линий, связанных с классовым блоком и общим ромбиком. Для представления однонаправленной ассоциации на линии изображается стрелка, указывающая направление.

Так же ассоциация может быть задана именем и в данном случае на концах соответствующей линии будут отображены свойства: индикаторы, принадлежность, мультипликаторы и т.д. [2].

Агрегация

Агрегацией является разновидность ассоциации, которая определяет отношение между целым и его частями. Агрегация не может определять отношение более двух классов (контейнер и содержимое). Агрегация так же может быть иметь имя. В языке UML агрегация отображается в виде пустого ромбика на блоке класса и линией, связывающей этот ромбик с содержащимся классом. На рисунке 3.6. представлен пример агрегации, отображающей связь между классами «Professor» и «Class».

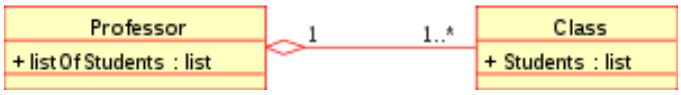

Рис.3.6. Диаграмма классов, показывающая агрегацию между двумя классами

Агрегация отображает отношение, когда один класс является коллекцией других классов. По умолчанию, агрегации присуще наименование «агрегация по ссылке», что означает отсутствие зависимости времени существования содержащихся классов от содержащего их класса. Другими словами, если содержащий класс будет уничтожен, то его содержимое – нет.

Композиция

Композиция является более строгим вариантом агрегации. Композицию так же называют агрегацией по значению. В отличие от агрегации композиции соответствует жесткая зависимость времени существования содержащихся классов от содержащего их класса. Если содержащий класс будет уничтожен, то так же будет уничтожено и все его содержимое. Композиция отображается в виде закрашенного ромбика на блоке класса и линией, связывающей этот ромбик с содержащимся классом.

Различия между композицией и агрегацией

В композиции должен использоваться мультипликатор «0..1» или «1», это отображает, что данная часть является частью только одного целого. В отличие от композиции в агрегации может использоваться любой мультипликатор.

В качестве наглядного примера приведем отношение. Квартира является частью дома – это композиция, так как квартира без дома существовать не может.

Квартира содержит мебель – это агрегация, так как мебель не является неотъемлемой частью квартиры и квартира может существовать без мебели.

Обобщение (наследование)

Для отображения связи между классом-родителем и классом-потомком в языке UML используется **обобщение**. Обобщение вводится в диаграмму классов, если в системе существуют несколько классов , имеющих схожее поведение (общие элементы поведения связываются с более высоким уровнем, образуя класс-родитель) (рис. 3.7).

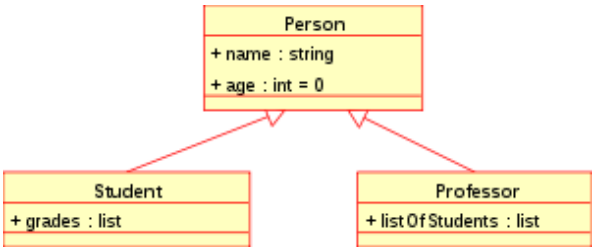

Рис. 3.7. Диаграмма классов, показывающая обобщение

Например: животные являются суперклассом млекопитающих. Млекопитающие являются суперклассом приматов и так далее. Эту взаимосвязь можно описать фразой «А – это Б» (приматы являются млекопитающими, млекопитающие являются животными). Так же обобщение называют как наследование или «is a» взаимосвязь.

Обобщение отображается в виде линии с пустым треугольником у суперкласса (рис. 3.8).

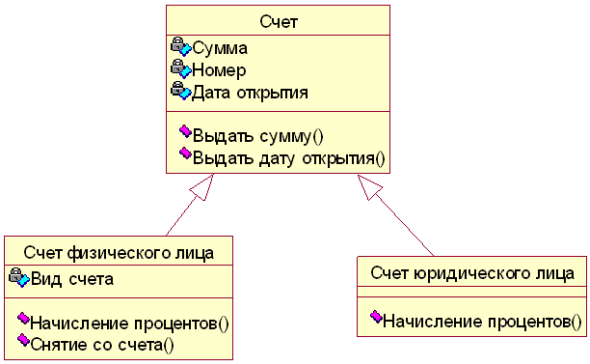

Рис.3.8. Обобщение

В языке UML существует возможность построения моделей, обладающих различным уровнем детализации. На рис. 3.9 показана детализация модели набора товаров, которая была представлена на рис. 3.5.

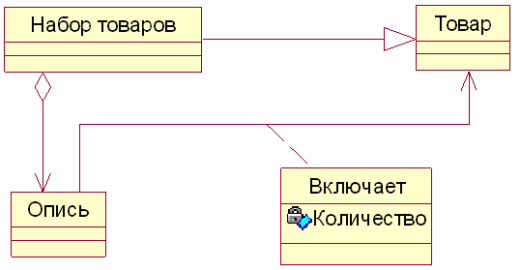

Рис. 3.9. Детализация модели набора товаров

С помощью обобщения показывается, что набор товаров так же является товаром, который может быть предметом продажи, поставки, заказа и т.д. На диаграмме содержится класс «Опись», в который содержит список товаров, входящих в набор. Так же содержится класс-ассоциация «Включает», который определяет количество каждого вида товаров в наборе [2].

Реализация

С помощью **реализации** задается отношение между двумя классами модели, где один класс (*клиент*) реализует поведение, которое задается другим классом (*поставщик*). Реализация отображается аналогично наследованию, только с пунктирной линией.

Зависимость

С помощью **зависимости** задается отношение между двумя классами модели, при котором изменение одного класса приводит к изменению другого класса, причем обратная зависимость не является обязательной. Данное отношение возникает, когда объект выражается в виде параметра или локальной переменной.

Зависимость отображается в виде пунктирной линии со стрелкой, связывающей зависимый элемент с тем, от которого он зависит. Зависимость может выражаться между экземплярами, классами или экземпляром и классом.

Уточнения отношений

Уточнение имеет отношение к уровню детализации. Один пакет является уточнением другого, если он содержит те же элементы, но представленные в более подробном виде. Например, при написании книги вначале можно использовать предложения, сформулированные таким образом, что в них кратко описывается содержание каждой главы. Предположим, что резюме, соответствующее каждой главе в виде отдельного элемента входит в пакет «Предложение». В пакет «Завершённая книга» входят элементы, являющиеся законченными главами. В данном случае пакет «Завершённая книга» является уточнением пакета «Предложение».

Мощность отношений (Кратность)

Мощностью отношения (мулитипликатор) называется число связей между каждым экземпляром класса в начале линии с экземпляром класса в ее конце. В таблице 1 представлены отношения и их интерпретация.

Таблица 1

Отношения между сущностями

Нотация	Объяснение	Пример
1	Обязательно один экземпляр	У собаки одна мать
0..1	Ноль или один экземпляр	Собака имеет или не имеет хозяина
1..*	Один или более экземпляров	У собаки есть хотя бы одно место, где она спит
0..* или *	Ноль или более экземпляров	У собаки может быть, а может и не быть щенков

3.3 Работа в StarUML

Создание нового проекта:

1. Запустите программу StarUML на рабочем столе. 
2. Создание нового проекта: выберите меню **File → New Project.**

Создание диаграмм

Палитра элементов содержит различные типы элементов, доступных для создания в зависимости от типа диаграммы (рис. 3.10). Список доступных элементов изменяется при переходе от диаграммы одного типа к диаграмме другого типа.

1.Выберите тип создаваемого элемента на палитре элементов.

2.Щёлкните желаемое место для нового элемента на диаграмме, чтобы создать там элемент.

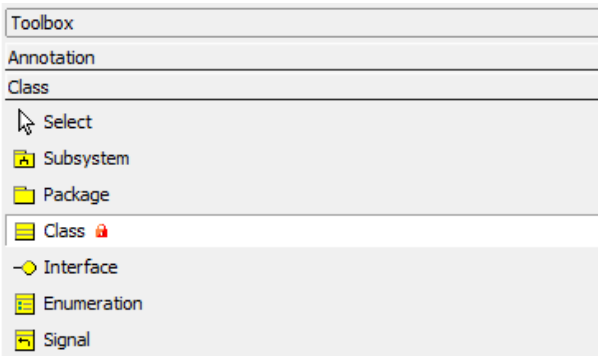

Рис.3.10. Палитра элементов

Что бы редактировать созданный элемент на диаграмме, нажмите в палитре инструментов стрелочку «select» (рис. 3.11). После этого вы можете выбирать элементы, изменять их имя (двойным щелчком мыши), размер и местоположение, менять их свойства (правым щелчком мыши).

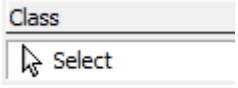

Рис.3.11. Режим редактирования элементов

Все остальные манипуляции с элементами можно производить, так же как и в любых других программах, копирование, вставка, выбор сразу нескольких элементов (при удерживании нажатой клавиши «shift») и т.д.

Создание подсистемы

1.Выберите в палитре элементов «Subsystem».

2. Затем щелкните место или границу, куда нужно поместить подсистему.

3.Сразу после создания подсистемы на диаграмме классов (справа) будет открыт её горячий диалог. В горячем диалоге, введите имя подсистемы.

Создание интерфейса подсистемы

1.Таким же образом создайте элемент «Interface».

2.Щелкните кнопку **Toolbox → Realization.**

3.Проведите линию от подсистемы к интерфейсу.

4.Между интерфейсом и подсистемой будет создано отношение реализации интерфейса (рис. 3.12).

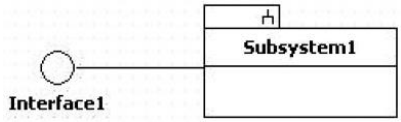

Рис.3.12. Отношение между подсистемой и интерфейсом

Добавление операции к подсистеме

Нажмите правой кнопкой мыши на подсистему и в контекстном меню выберите **Add → Operation** (рис. 3.13).

Рис. 3.13. Создание операции

Или двойным щелчком левой кнопки мыши на элементе можете создавать новые операции и атрибуты или удалять их, путём нажатия «+» и «-» (рис. 3.14).

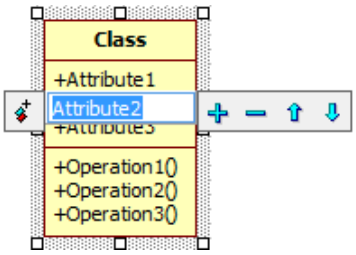

Рис. 3.14. Создание операций и атрибутов

Для изменения свойства видимости атрибута/операции необходимо воспользоваться контекстным меню, расположенным слева (рис. 3.15).

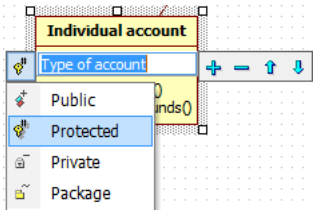

Рис. 3.15. Изменение свойства видимости атрибута

Создание класса

Таким же образом создайте класс, выбрав элемент «Class» на панели элементов. И добавьте ему атрибут. Таким же образом можно добавлять операции классу.

Добавление параметра к операции

Выберите операцию в навигаторе модели (**View → Model Explorer**), выберите пункт **Add → Parameter** в контекстном меню, новый параметр будет добавлен.

Чтобы переместить атрибут или операцию в другой класс, щёлкните атрибут (или операция) в навигаторе модели и перетащите его в другой класс [3].

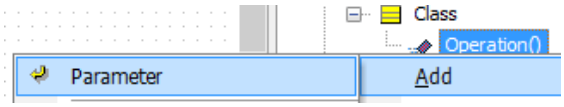

Рис.3.16. Создание параметров

3.4 Проектирование приложения

Одним из возможных подходов при проектировании приложения в целом, и отдельных классов в частности, является лексический анализ текса. В тексте выделяются имена существительные и глаголы. Имена существительные являются кандидатами на классы-сущности и их методы (в зависимости от того, что от чего зависит). Глаголы являются кандидатами на методы.

Для иллюстрации данного подхода рассмотрим конкретный пример задания.

Вооружить **подразделение оружием**, *определить* его **общий вес** и **стоимость**.

Слова, выделенные сплошным подчеркиванием, станут классами-сущностями, так как они являются основными понятиями в данном контексте. Вес и стоимость станут атрибутом класса *Оружия*, так как они являются его характеристикой. Также можно судить, что *Подразделение* будет содержать множество объектов класса *Оружие.* Глагол *определить* станет двумя методами в приложении, позволяющими рассчитать вес и стоимость *Оружия* в *Подразделении*. Также резонно полагать, что эти методы будут иметь дело с *Подразделением*. Иначе говоря, методы *Определить вес* и *Определить стоимость* будут принимать объект класса *Подразделения* на вход, то есть мы уже можем определить сигнатуру этих методов. Дополнительно можно спроектировать еще один класс – *Логика подразделения,* который будет содержать в себе два описанных метода.

Таким образом, подход на основе лексического анализа текста задания позволяет нам эффективно проектировать приложение.

3.5 Создание DLL

Динамически подключаемая библиотека (DLL) представляет собой исполняемый код, однако он не содержит функции Main. Вызов методов происходит из exe-сборки основного проекта. Плюсом использования DLL является то, что в память DLL загружается только при вызове методов находящихся в ней. Для того чтобы создать динамически подключаемую библиотеку в среде Visual Studio, необходимо выбрать **New Project →Class Library.**

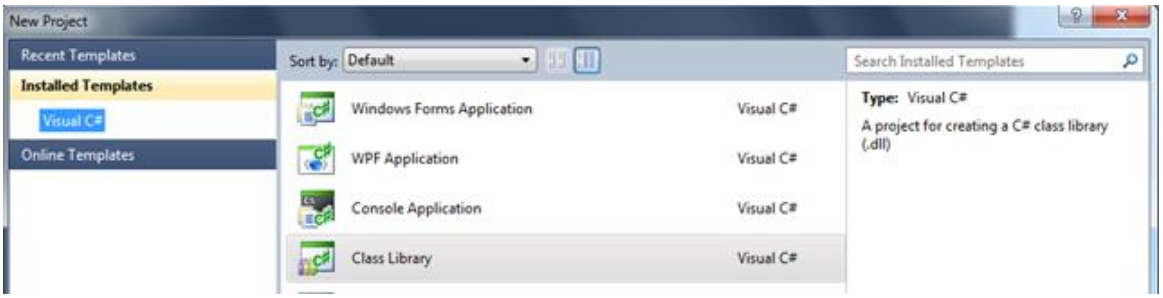

Рис. 3.24. Тип проекта

3.6 Соглашение о кодировании

Под соглашением о кодировании подразумевается набор соглашений о том, каким образом писать программный код. Как правило, соглашение о кодировании включает: как называть переменные различных типов, как именовать классы, какой отступ использовать, как оформлять операторы и пр. Также соглашение о кодировании может включать запреты, например на использование оператора *goto*, что сейчас считается плохим тоном – это усложняет чтение кода и его отладку.

В целом, соглашение о кодирование позволяет разработчикам более эффективно читать и сопровождать программный код (вносить в него изменения). Соглашение о кодирование позволяет сосредоточиться не на том, *как* написан программный код, а на то, *что* в нем написано. В отсутствие оного, разработчику пришлось бы сначала привыкнуть к стилю автора данного кода (разработка на крупных проектах ведется командой), и только потом он сможет уже сосредоточиться на понимании кода. Соглашение о кодировании приниматься, как правило, для программного проекта, и позволять унифицировать и стандартизировать программный код – привести его к единообразию.

В данном курсе мы будем придерживаться следующего соглашения о кодировании (и требований к программному коду) (применительно к языку C#):

1. Имена переменных, классов, проектов должны носить осмысленные названия. Запрещается использовать имена типа *a, b, Form1, button1*. Вместо этого необходимо использовать такие имена как *count, price, MainForm, submitButton*;

2. Имена переменных должны начинаться с маленькой буквы;

3. Имена классов должны начинаться с большой буквы;

4. В название классов/переменных каждое последующее слово должно писаться слитно с предыдущим и с большой буквы (CamelStyle);

5. Название класса должно являться именем существительным;

6. Название метода должно начинаться с большой буквы;

7. Название метода должно носить характер действия и являться глаголом или содержать его, например *GetPrice;*

8. Имена полей класса с модификатором доступа *private* должны начинаться со знака «_»;

9. Атрибуты класса (Property) должны начинаться с большой буквы;

10. Запрещается использовать цифры в названиях, если они не носят осмысленный характер. Нельзя: *form1*, можно: *ak47*;

11. Названия абстрактных классов должны начинаться со слова *Abstract*, например *AbstractWeapon*;

12. Название интерфейса должно начинаться с буквы *I*, например *IService*;

13. Имена переменных обозначающих элементы графического пользовательского интерфейса в своем названии должны содержать название типа (в конце). Например: *submitButton;*

14. Каждый класс должен создаваться в отдельном файле, имя которого должно совпадать с названием класса;

15. В одной строке должна объявляться одна переменная.

Нельзя:

```
int count, price = 0;
```

Можно:

```
int count = 0;
int price = 0;
```

Задание

Необходимо описать иерархию классов предметной области в виде UML диаграммы и классов на языке C# в соответствии с выбранным вариантом задания (см. приложение).

Необходимо выполнить только первую часть задания, т.е. логику по расчету значений и вывод данных на экран выполнять не надо, только описание классов.

Требования и рекомендации:

1. Иерархия классов должна состоять минимум из 3 уровней;
2. Каждый класс должен содержать уникальный набор атрибутов;
3. Классы должны быть оформлены в виде DLL библиотеки в консольном проекте;
4. При именовании переменных и описании классов необходимо придерживаться *соглашения о кодировании* (см. п. 3.6).

Пример описания класса предметной области на языке C#:

```
public class Product
{
    private string _name;
    private int _price;
    private double _weight;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }

    public int Price
    {
        get { return _price; }
        set { _price = value; }
    }

    public double Weight
    {
        get { return _weight; }
        set { _weight = value; }
    }
}
```

Пример диаграммы классов:

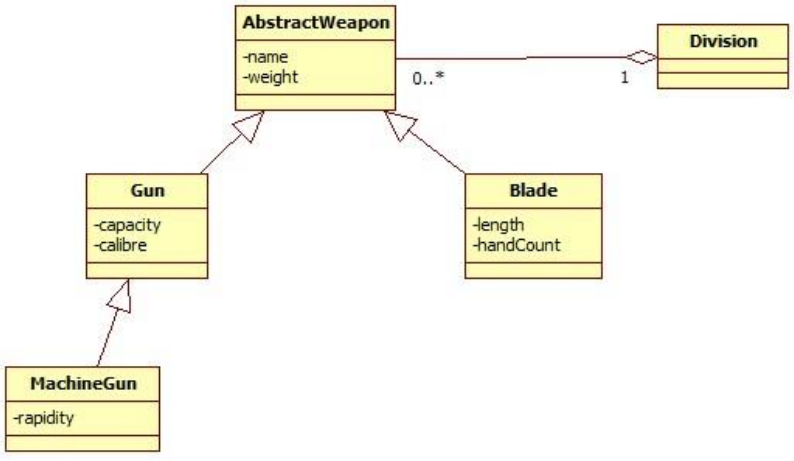

Рис.3.25. Пример диаграммы классов предметной области

Варианты заданий

1. **Цветочница.** Определить иерархию цветов. Создать несколько объектов-цветов. Собрать букет с определением его стоимости.

2. **Новогодний подарок.** Определить иерархию конфет и прочих сладостей. Создать несколько объектов-конфет. Собрать детский подарок с определением его веса.

3. **Электрик.** Определить иерархию электроприборов. Включить некоторые в розетку. Посчитать потребляемую мощность

4. **Диета.** Определить иерархию овощей. Собрать в салат. Посчитать калорийность.

5. **Меломан.** Определить иерархию музыкальных композиций. Записать на диск сборку. Посчитать продолжительность.

6. **Камни.** Определить иерархию драгоценных и полудрагоценных камней. Отобрать камни для ожерелья. Посчитать общий вес (в каратах) и стоимость.

7. **Оружие.** Определить иерархию оружия (холодного и огнестрельного). Вооружить военное подразделение. Посчитать стоимость.

8. **Транспорт.** Определить иерархию пассажирского транспорта. Создать набор маршрутов для перемещения и точки А в точку Б. Посчитать общую стоимость проезда.

9. **Авиакомпания.** Определить иерархию самолетов. Создать авиакомпанию. Посчитать общую вместимость (в пассажирах).

10. **Автомобили.** Определить иерархию легковых автомобилей. Создать автопарк организации. Посчитать стоимость автопарка.

Контрольные вопросы

1. Что такое наследование?
2. Что такое инкапсуляция?
3. Что такое полиморфизм?
4. Что такое абстрактный класс?

Список литературы

1. Блинов И.М., Романчик В.С. Java. Промышленное программирование: практ. пособие – Минск: УниверсалПресс, 2007. – 704 с.
2. Диаграммы классов UML. Логическое моделирование [Электронный ресурс]. – Режим доступа: http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=3, свободный.
3. StarUML. Руководство пользователя [Электронный ресурс]. – Режим доступа: http://staruml.sourceforge.net/docs/user-guide(ru)/user-guide.pdf, свободный.
4. Джим Арлоу, Айла Нейштадт UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, Символ-Плюс, 2007 – 624 с.
5. Дж. Рамбо, М. Блаха UML 2.0. Объектно-ориентированное моделирование и разработка, Питер, 2007 – 544 с.

Лабораторная работа № 4
Создание логики приложения

Цель работы:

Познакомится с разбиением приложения на слои. Освоить основные принципы объектно-ориентированного проектирования. Научиться применять шаблон Factory Method.

Введение

Программное приложение должно разрабатывается с учетом определенных принципов, таких как отделение логики от представления, отделение логики от данных, высокое зацепление [за обязанности] (High Cohesion), низкое связывание (Low Coupling). Также на уровне детального проектирования целесообразно применять шаблоны проектирования: порождающие, структурные и поведенческие. Одним из таких шаблонов проектирования является *FactoryMethod*. Данная лабораторная работа затрагивает все вышеперечисленные аспекты и позволит студенту познакомиться с ними на практике.

4.1. Разделение приложения на слои

Концепция слоев (layers) – одна из общеупотребительных моделей, используемых разработчиками программного обеспечения для разделения сложных систем на более простые части.

В таблице 2 представлены основные виды слоёв и их функции (согласно М. Фаулеру [1]).

Таблица 2

Основные виды слоёв

Слой	Функции
Представление	Предоставление услуг, отображение данных, обработка событий пользовательского интерфейса (щелчков кнопками мыши и нажатий клавиш), обслуживание запросов HTTP, поддержка функций командной строки и API пакетного выполнения
Домен	Бизнес-логика приложения
Источник данных	Обращение к базе данных, обмен сообщениями, управление транзакциями и т.д.

Слой представления охватывает все, что имеет отношение к общению пользователя с системой. Он может быть настолько простым, как командная строка или текстовое меню, но сегодня пользователю, вероятнее всего, придется иметь дело с графическим интерфейсом, оформленным в стиле толстого клиента (Windows, Swing и т.п.) или основанным на HTML. К главным функциям слоя представления относятся отображение информации и интерпретация вводимых пользователем команд с преобразованием их в соответствующие операции в контексте домена (бизнес-логики) и источника данных.

Источник данных – это подмножество функций, обеспечивающих взаимодействие со сторонними системами, которые выполняют задания в интересах приложения. Код этой категории несет ответственность за мониторинг транзакций,

управление другими приложениями, обмен сообщениями и т.д. Для большинства корпоративных приложений основная часть логики источника данных сосредоточена в коде СУБД.

Логика домена (бизнес-логика или логика предметной области) описывает основные функции приложения, предназначенные для достижения поставленной перед ним цели. К таким функциям относятся вычисления на основе вводимых и хранимых данных, проверка всех элементов данных и обработка команд, поступающих от слоя представления, а также передача информации слою источника данных.

Помимо необходимости разделения на слои, существует правило, касающееся взаимоотношения слоев: зависимость бизнес-логики и источника данных от уровня представления не допускается. Другими словами, в тексте приложения не должно быть вызовов функций представления из кода бизнес-логики или источника данных. Правило позволяет упростить возможность адаптации слоя представления или замены его альтернативным вариантом с сохранением основы приложения. Связь между бизнес-логикой и источником данных, однако, не столь однозначна и во многом определяется выбором типовых решений для архитектуры источника данных [1].

4.2. Детальное проектирование

Интерфейс

Интерфейс (от лат. *Inter* – «между», и *face* – «поверхность») – семантическая и синтаксическая конструкция в коде программы, используемая для специфицирования услуг, предоставляемых классом или компонентом. Интерфейс определяет границу взаимодействия между классами или компонентами, специфицируя определенную абстракцию, которую осуществляет реализующая сторона.

Другими словами, интерфейс – это набор открытых методов, которые определяют поведение класса. Класс, реализующий интерфейс берет на себя обязанность определить методы интерфейса.

Интерфейс класса должен составлять согласованную абстракцию. Иначе говоря, открытые методы класса должны быть логически согласованны между собой, что соответствует принципу проектирования High Cohesion (Сильное зацепление).

Два принципа проектирования

High Cohesion (высокое сцепление). Сцепление – это мера сфокусированности класса на своих задачах. При большом сцеплении код получается более читабельным и простым для повторного использования. В лучшем случае класс должен иметь лишь одну задачу (принцип единственной ответственности).

Класс с низкой степенью зацепления выполняет много разнородных функций или несвязанных между собой обязанностей. Такие классы создавать нежелательно, поскольку они приводят к возникновению следующих проблем:

- трудность понимания;
- сложность при повторном использовании;
- сложность поддержки;
- ненадежность, постоянная подверженность изменениям.

Классы с низкой степенью зацепления, как правило, являются слишком «абстрактными» или выполняют обязанности, которые можно легко распределить между другими объектами.

Low Coupling (низкое связывание). Связность – это мера зависимости между классами или модулями. При слабой связности модули взаимодействуют через устойчивый интерфейс и не зависят от реализации друг друга.

При высокой связности проявляются проблемы следующего плана:

- изменения в одном модуле тянут за собой изменения в другом;
- тяжело читать и понимать код отдельного модуля;
- модуль тяжело заново использовать и тестировать.

Связность является объективной величиной, ее можно вычислить и зависит от таких параметров как количество используемых глобальных переменных, количество передаваемых параметров и т.п. Величина связности изменяется от 0 до 1. При нулевой связности модули не взаимодействуют вовсе.

Шаблон Factory Method

Factory Method – порождающий шаблон проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, Factory делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне (рис. 4.1).

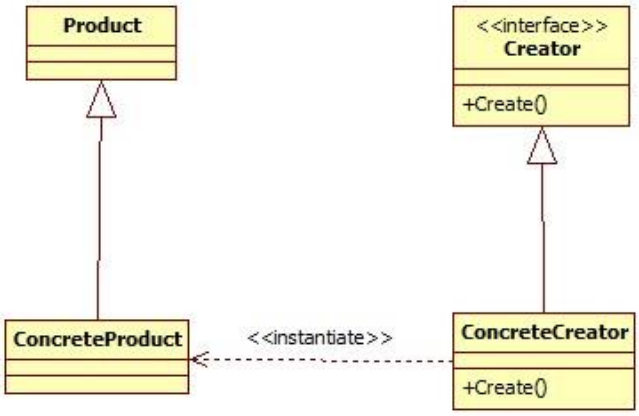

Рис. 4.1. UML диаграмма шаблона Factory Method

Рефакторинг

Под *рефакторингом* понимается последовательное, итеративное изменение (улучшение) программного кода без изменения его функциональности. При написании кода его не всегда удается написать правильно, в смысле соблюдения принципов проектирования и обеспечения удобства его последующей модификации и читабельности. В таких случаях считается допустимым написать рабочий код с некоторыми архитектурами (и не только) нарушениями. Например, вы можете написать логику расчета стоимости в классе, отвечающем за пользовательский ин-

терфейс (класс Формы), однако затем применить последовательный рефакторинг (по мере необходимости): вынести логику расчета в отдельный метод в этом классе, вынести этот метод в отдельный класс в этом проекте, вынести этот класс в другой проект.

Разберем пример последовательного рефакторинга.

Предположим, вы создали несколько объектов классов-сущностей в классе Program:

```
class Program
{
    static void Main(string[] args)
    {
        //создание объектов
        Gun gun = new Gun();
        gun.Name = "TT";
        gun.Price = 10000;
        gun.Calibre = 5.45;
        gun.Capacity = 8;

        Blade blade = new Blade();
        blade.Name = "Штык-нож";
        blade.Price = 1500;
        blade.Lenght = 20;
        blade.HandCount = HandCount.One;

        //добавление оружия в "подразделение"
        Division division = new Division();
        division.AddWeapon(gun);
        division.AddWeapon(blade);
     }
 }
```

Затем вы вынесли логику создания объектов в отдельный метод:

```
class Program
{

    static void Main(string[] args)
    {
        Division division = CreateDivision();
    }

    private static Division CreateDivision()
    {
        //создание объектов
        Gun gun = new Gun();
        gun.Name = "TT";
        gun.Price = 10000;
        gun.Calibre = 5.45;
        gun.Capacity = 8;

        Blade blade = new Blade();
        blade.Name = "Штык-нож";
        blade.Price = 1500;
        blade.Lenght = 20;
        blade.HandCount = HandCount.One;
```

```
        //добавление оружия в "подразделение"
        Division division = new Division();
        division.AddWeapon(gun);
        division.AddWeapon(blade);

        return division;
    }
}
```

Это позволит сосредоточиться в методе *Main* на логике приложения, а не на том, как она реализована. Основная логика в методе *Main* заключается в создании «подразделения», а то каким образом оно создается – это уже детали, которые можно посмотреть, зайдя в метод *CreateDivision*. Такой подход упрощает понимание приложения и имеет ряд преимуществ. Так как человек не может удерживать в своем внимании более 5-7-ми объектов, то многострочный код скрывает, то с какой целью был написан этот код.

Вынося код с *одной функциональностью* (метод должен выполнять не более одной функции) в отдельный метод мы позволим программисту сосредоточиться на цели, а не на коде. Дополнительно, такой подход является самодокументируемым, то есть участок кода получил комментарий в виде названия метода, и нам не пришлось писать дополнительный комментарий. В идеале метод *Main* должен содержать несколько вызовов таких методов, что позволит без труда понять логику его работы. В противном случае метод *Main* содержал бы сотню строк кода (вместо вызова семи методов), что не позволило бы нам понять его логику, по крайней мере, сразу.

Как было сказано выше, метод должен выполнять одну и только одну функцию, помимо этого название метода должно отражать эту функцию. Если метод называется *ReadFile*, то он не может вносить изменения в файл. Такое бы поведение метода вызвало бы конфуз у программиста, так как он не ожидает такого поведения от метода.

Затем вы вынесли логику создания объектов в отдельный класс:

```
class Program
{

    static void Main(string[] args)
    {
        Division division = DivisionFactory.CreateDivision();
    }
}

public class DivisionFactory
{
    public static Division CreateDivision()
    {
        //создание объектов
        Gun gun = new Gun();
        gun.Name = "TT";
        gun.Price = 10000;
        gun.Calibre = 5.45;
        gun.Capacity = 8;
```

```
            Blade blade = new Blade();
            blade.Name = "Штык-нож";
            blade.Price = 1500;
            blade.Lenght = 20;
            blade.HandCount = HandCount.One;

            //добавление оружия в "подразделение"
            Division division = new Division();
            division.AddWeapon(gun);
            division.AddWeapon(blade);

            return division;
        }
    }
```

Вынесение метода в отдельный класс в данном случае следует из принципа *High Cohesion*, то есть высокое зацепление за обязанности. Класс обязанностью которого является запуск приложения не должен содержать логику приложения, а должен делегировать ее другим классам.

Данный пример наглядно демонстрирует последовательное применение рефакторинга, а также использования шаблона *Factory Method.* Правда шаблон *Factory Method* применяется не в чистом виде, а его вырожденный и гибридный случай: отсутствует иерархия классов-сущностей – только один класс *Division,* и соответственно отсутствует иерархия классов *Factory* – один класс со статическим методом; в методе помимо создания объекта идет инициализация его полей, что больше походит на шаблон *Builder.* Однако в рамках данной лабораторной работы такой пример считается приемлемым, который демонстрирует обособление создания объекта в другой класс.

Задание

Для созданных классов предметной области в предыдущей лабораторной работе необходимо создать логику приложения в соответствии с вариантом задания (см. приложение). При выполнении лабораторной работы необходимо соблюдать следующие правила:

1. Необходимо создавать логику приложения в отдельных классах;
2. Классы логики приложения не должны зависеть от интерфейса пользователя, их необходимо выделить в отдельный пакет либо библиотеку классов;
3. Для использования логики приложения в интерфейсе пользователя необходимо вызывать методы классов, содержащих логику.

Пример:

```
public class ProductCalculator
{

      public int GetTotalPrice(Product product)
      {
            int totalPrice;
            //выполнение необходимых расчетов
            …
            return totalPrice;
      }
}

class Program
{

      static void Main(string[] args)
      {
            Product = new Product();

            //заполнение «продукта» данными
            …

            //получение стоимости
            ProductCalculator calculator = new ProductCalculator();
            int price = calculator.GetTotalPrice(product);
      }
}
```

Для создания объектов предметной области необходимо использовать шаблон Factory Method.

Пример:

```
public class ProductFactory
{

      public Product CreateProduct ()
      {
            Product product= new Product();
            //инициализация данных объекта
```

```
            …
            return product;
        }
    }
```

Программу необходимо реализовать в виде консольного приложения на языке C#. В соответствие со следующими правилами:

1. Необходимо выделить логику приложения и уровень представления (интерфейс пользователя) в отдельные проекты. Рекомендуются оформить логику приложения в виде библиотеки классов.

Пример добавление ссылки на другой проект:

Для того чтобы использовать классы, расположенные в другом проекте (в dll библиотеке), необходимо добавить на него ссылку в проект, в котором будут использоваться эти классы. Для добавления ссылки необходимо нажать правой кнопкой мыши на проект и выбрать *Add Reference…* (рис. 4.2).

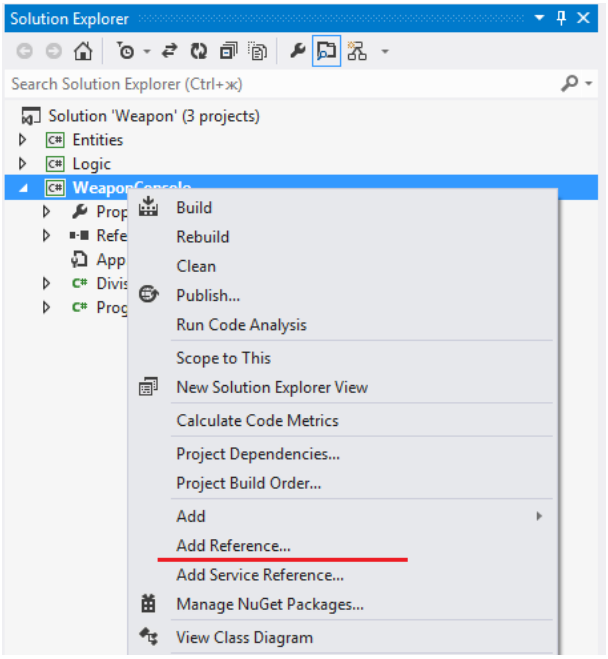

Рис. 4.2 Добавление ссылки на проект

Затем выбрать проекты, на которые необходимо добавить ссылку (те в которых содержаться необходимые классы) (рис. 4.3).

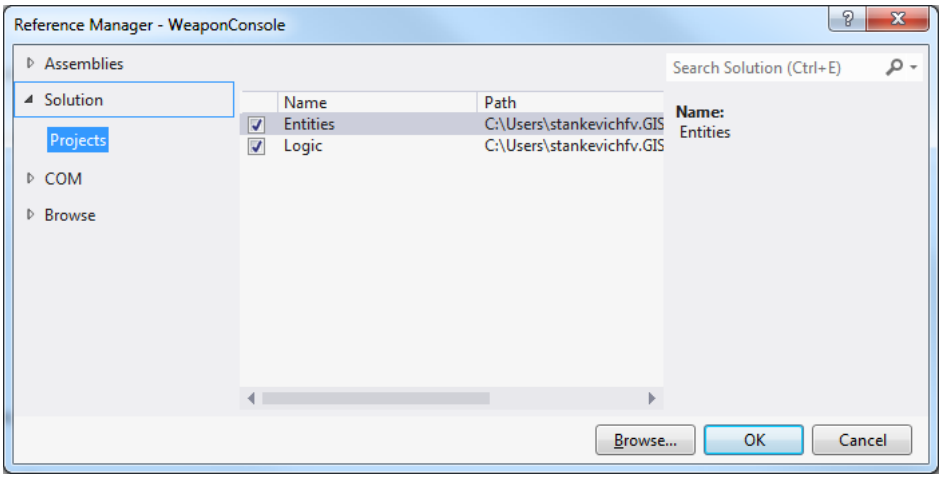

Рис. 4.3 Выбор проектов

2. Вывод данных на экран должен быть реализован в отдельных классах.

Пример:

```
public class ProductPrinter
{

	public void Print(Product product)
	{
		//вывод данных экран
		Console.WrileLine("Название = " + product.Name);
		…
	}
}

class Program
{

	static void Main(string[] args)
	{
		Product = new Product();

		//заполнение «продукта» данными
		…

		//вывод данных
		ProductPrinter printer = new ProductPrinter();
		printer.Print(product);
	}
}
```

3. Точка входа приложения (метод Main) должна только лишь делегировать (перенаправлять) вызовы в другие классы (интерфейса пользователя, логики).

```
class Program
{

	static void Main(string[] args)
	{
```

```
            Division division = DivisionFactory.CreateDivision();

            DivisionPrinter printer = new DivisionPrinter();
            printer.Print(division);

            DivisionCalculator calculator = new DivisionCalculator();
            int price = calculator.GetPrice(division);

            Console.WriteLine("Price = " + price);

            Console.ReadKey();
      }
}
```

Пример также должно выглядеть и ваше приложение.

4. В приложении необходимо реализовать логирование (информационно-отладочный вывод) посредством реализации интерфейса ILogger. Должно быть, как минимум две реализации, например, логгирование на консоль и в файл. Создание конкретной реализации должно быть выполнено с помощью шаблона Factory Method. *Это задание является опциональным.*

Пример:

```
public interface ILogger
{
      void Log (string message);
}

public class ConsoleLogger : ILogger
{
      public void Log (string message)
      {
            //реализация
            …
      }
}

public class FileLogger : ILogger
{
      public void Log (string message)
      {
            //реализация
            …
      }
}

public class LoggerFactory
{
      public ILogger CreateLogger(int type)
      {
            //реализация
            …
      }
}
```

Контрольные вопросы

1. Принципы разбиения приложения на модули.
2. Что такое интерфейс?
3. Отличие абстрактного класса от интерфейса.
4. Два принципа проектирования.
5. Шаблон Factory Method.

Список литературы

1. Фаулер М. Архитектура корпоративных приложений, Вильямс, 2006 г. – 544 с.

2. Блинов И.Н, Романчик В.С. Java 2. Практическое руководство. – Мн.: УниверсалПресс, 2005. – 400 с.

3. Блинов И.Н., Романчик В.С Java. Методы программирования, Четыре четверти, 2013 – 896 с.

4. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2007. – 266 с.