

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ  
Федеральное государственное бюджетное образовательное  
учреждение  
высшего профессионального образования  
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

---

УТВЕРЖДАЮ  
Директор ИК

\_\_\_\_\_ А.А. Захарова  
«\_\_» \_\_\_\_\_ 2015 г.

**Ф.В. Станкевич, Е.А. Мирошниченко,  
А.А. Друки, С.С. Пекарская**

**Под редакцией Ф.В. Станкевича**

**ОСНОВЫ ПРОМЫШЛЕННОГО ПРОГРАММИРОВАНИЯ**

Методические указания к выполнению лабораторных работ по  
курсу «Технологии программирования» для студентов II курса,  
обучающихся по направлению 230100 «Информатика и  
вычислительная техника»

Издательство  
Томского политехнического университета  
2015

УДК  
ББК  
номер

Авторы:

Ф.В. Станкевич, Е.А. Мирошниченко, А.А. Друки, С.С. Пекарская

Под редакцией Ф.В. Станкевича

Технологии программирования: методические указания к выполнению лабораторных работ по курсу «Технологии программирования» для студентов II курса, обучающихся по направлению 230100 «Информатика и вычислительная техника» / Ф.В. Станкевич, Е.А. Мирошниченко, А.А. Друки, С.С. Пекарская; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2015. – 73 с.

**УДК 00000**  
**ББК 00000**

Методические указания рассмотрены и рекомендованы  
к изданию методическим семинаром кафедры  
вычислительной техники  
«\_\_» \_\_\_\_\_ 2015 г.

Зав. кафедрой  
кандидат технических наук, профессор \_\_\_\_\_ *Н.Г. Марков*

Председатель учебно-методической  
комиссии \_\_\_\_\_ *А.Д. Чередов*

*Рецензент*

**Доктор технических наук, профессор**  
**Заведующий кафедрой «название»**  
**ФЮ**

© ФГБОУ ВПО НИ ТПУ, 2015  
© Ф.В. Станкевич, Е.А. Мирошниченко,  
А.А. Друки, С.С. Пекарская 2015  
© Оформление. Издательство Томского  
политехнического университета, 2015

## *О курсе*

В настоящее время объектно-ориентированное программирование (ООП) является стандартом для разработки крупных промышленных приложений, позволяющим эффективно организовывать программный код и управлять сложностью приложения. ООП обеспечивает расширяемость приложения, возможность разработки нового функционала и модернизацию существующего с наименьшими трудовыми затратами. Однако ООП неправильно рассматривать отдельно от сопутствующих ему технологий и принципов разработки программного кода.

Данный курс нацелен на обучение студентов именно таким технологиям и принципам, которые являются на сегодняшний момент неотъемлемой частью разработки крупных приложений наряду с ООП. 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 программирования (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 сцепление [за обязанности] (High Cohesion), шаблоны проектирования и другое.

Однако нередко, обучая студентов ООП, забывают о необходимости обучения вышеперечисленным принципам и технологиям, без которых немислима современная разработка программного обеспечения (ПО) на профессиональном уровне.

Курс «Основы промышленного программирования» обучает студентов использовать описанные принципы и технологии по средствам модельных задач. Такие задачи на первый взгляд могут показаться очень простыми, но приглядевшись к ним внимательнее, мы понимаем, что реализация такой задачи в соответствие с принципами ООП/ООД требует нетривиального решения (и творческого подхода), которое может вылиться в десятки классов.

Авторам курса видется, что именно такие модельные задачи позволят студентам наиболее быстро и безболезненно научиться применять такие технологии и принципы и на крупных проектах. Таким образом, авторы надеются, что студенты, придя на реальное производство, где занимаются профессиональной разработкой ПО, уже будет готовы к его реалиям.

*С уважением,  
Авторский коллектив*

# Лабораторная работа № 1

## Оформление текстового документа в Microsoft Office Word

### Цель работы:

Изучение и получение навыков работы с различными важными приёмами оформления текстовых документов в Microsoft Office Word.

### Введение

Умение работать с документацией является одной из важных навыков для профессионального разработчика программного обеспечения. Разработчику постоянно приходится сталкиваться с проектными документами: техническими заданиями, спецификацией требования к программному обеспечению и пр. Разработчик ПО должен обладать не только навыками по написанию кода, но и аналитическими. 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.

### 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.

#### 1.2 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 (описаны в разделе «**Ошибка! Источник ссылки не найден.**»). 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.

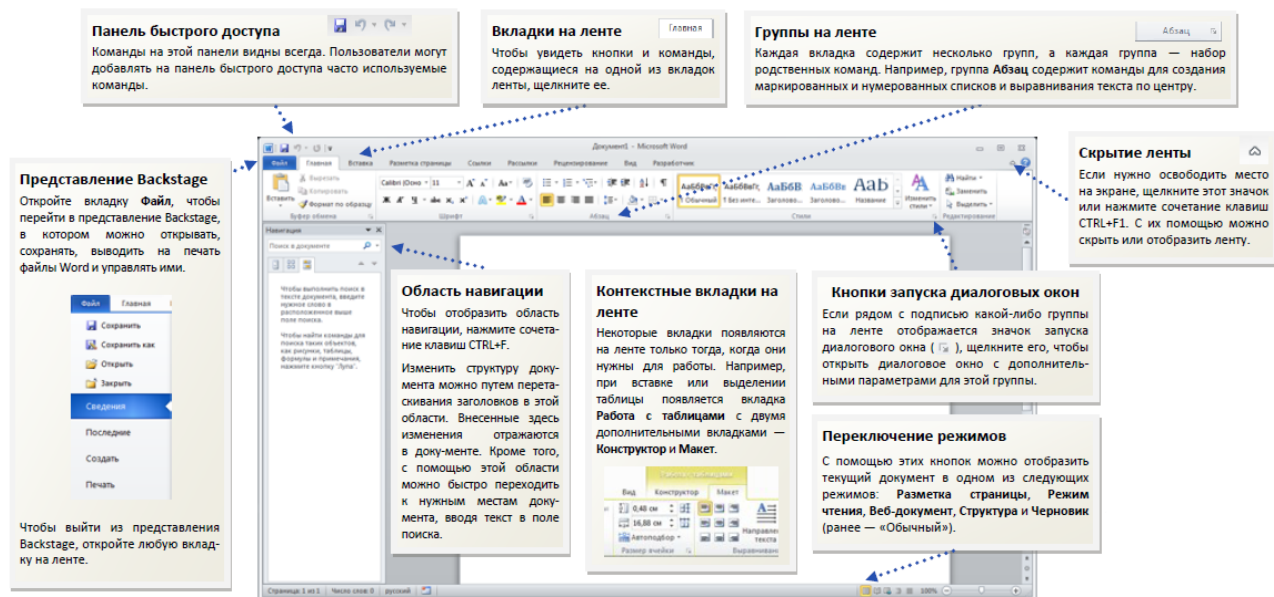


Рис. 1.1. Приложение Microsoft Office Word 2010

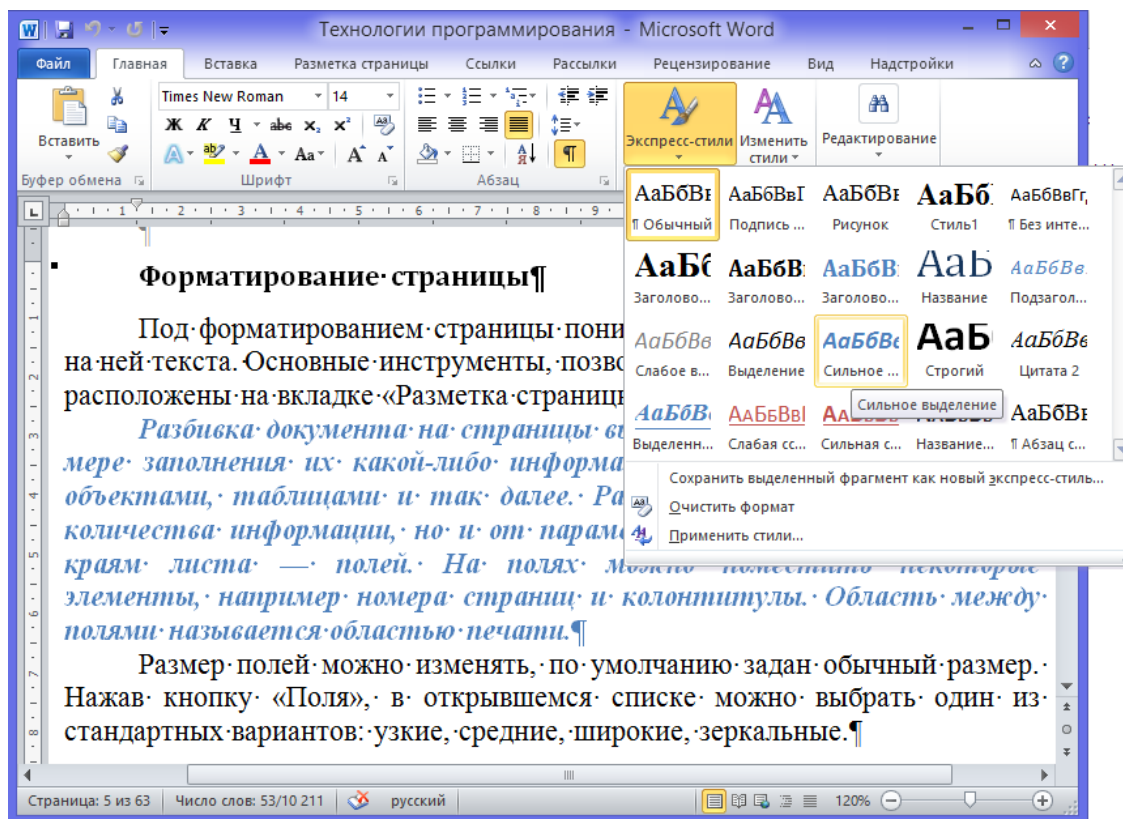


Рис. 1.2. Стили текста

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.

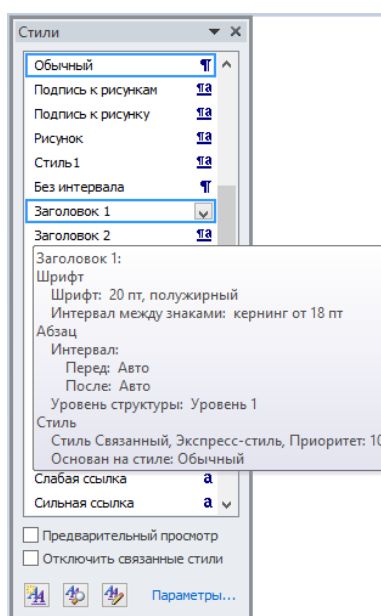


Рис. 1.3. Параметры стиля

При наведении курсора на название стиля справа появляется стрелочка, открывающая вспомогательное меню.

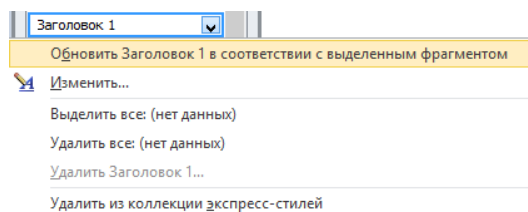



Рис. 1.4. Вспомогательное меню стиля

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

Чтобы создать новый стиль и задать ему необходимые параметры, необходимо воспользоваться кнопкой «Создать стиль» . В открывшемся диалоговом окне нужно ввести 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».

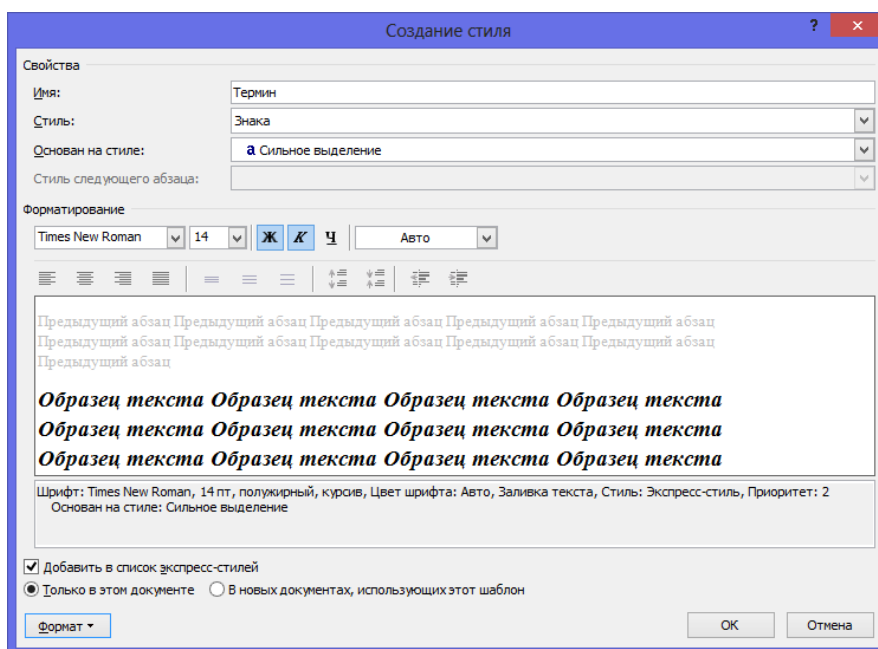


Рис. 1.5. Диалоговое окно «Создание стилей»

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.

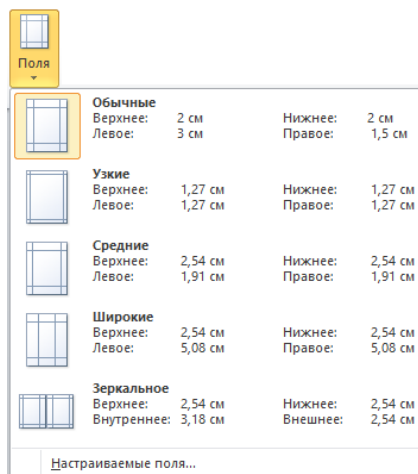


Рис. 1.6. Размеры полей документа

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.

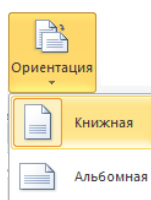


Рис. 1.7. Ориентация страницы

## 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  «Следующая сноска».

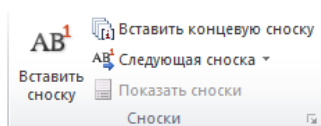


Рис. 1.7а. Вкладка «Ссылки»

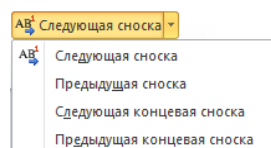


Рис. 1.7б. Меню «Следующая сноска»

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».

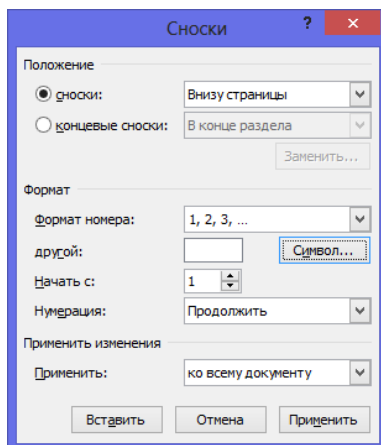


Рис. 1.8. Диалоговое окно «Сноски»

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.

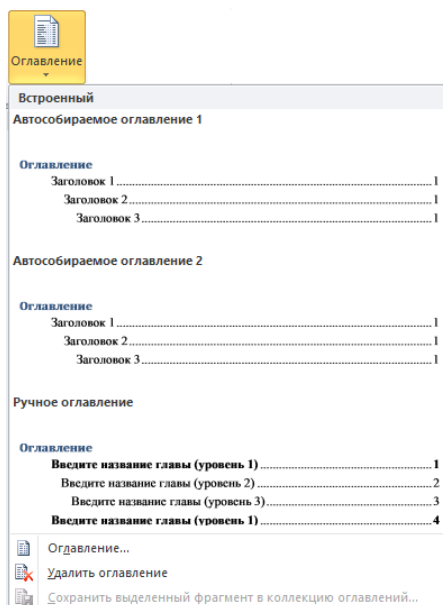


Рис. 1.9. Виды автоматически создаваемых оглавлений

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, нажать клавишу Ctrl и щёлкнуть 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 в 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.

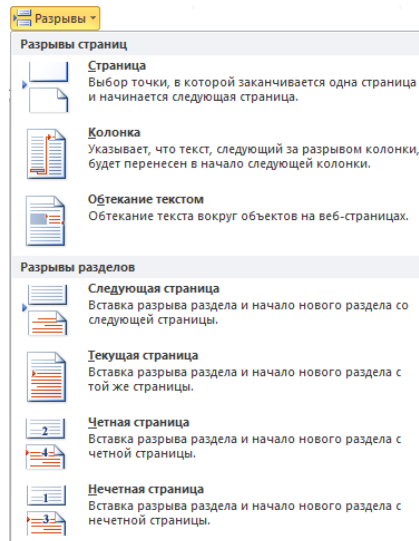


Рис. 1.10. Разрывы страниц/разделов

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.

### 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).

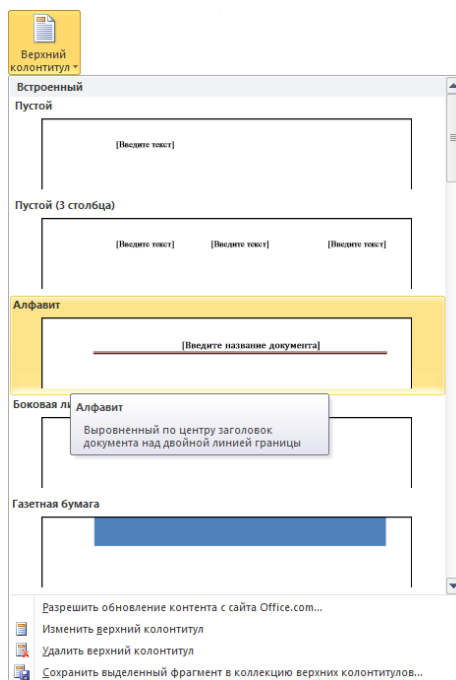


Рис. 1.12. Вставка колонтитула

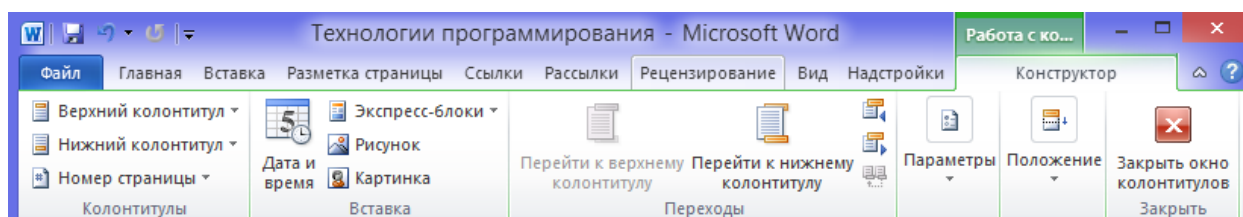


Рис. 1.13. Вкладка «Работа с колонтитулами»

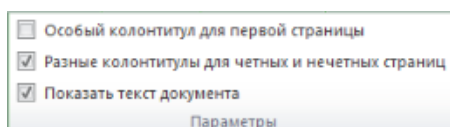


Рис. 1.14. Параметры колонтитулов

## 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).

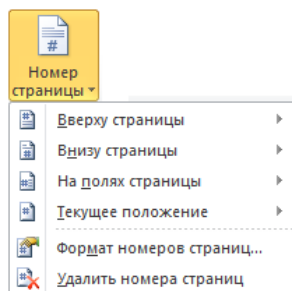


Рис. 1.15. Нумерация страниц

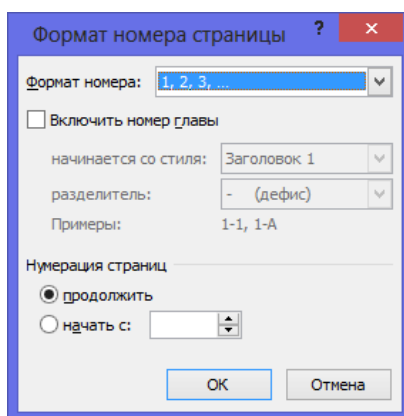


Рис. 1.16. Диалоговое окно «Формат номера страницы»

Для нумерации с учётом глав, необходимо в окне «Формат номера страницы» поставить флажок «Включить номер главы» определить с какого стиля начинаются главы, 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 щелчком 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».

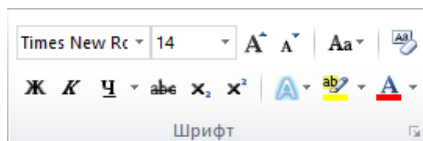


Рис. 1.17. Панель «Шрифт»

## 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.

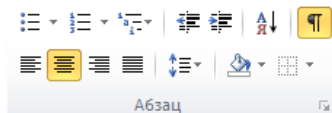





Рис. 1.18. Вкладка «Главная», панель «Абзац»

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.

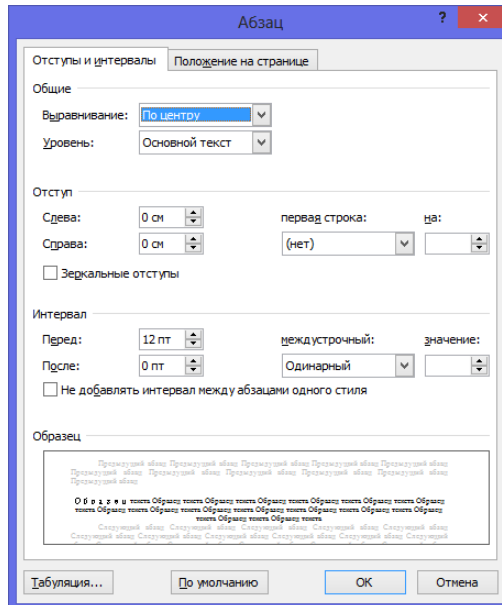


Рис. 1.19. Диалоговое окно «Абзац»

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 визуальнo вычлeнить 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».

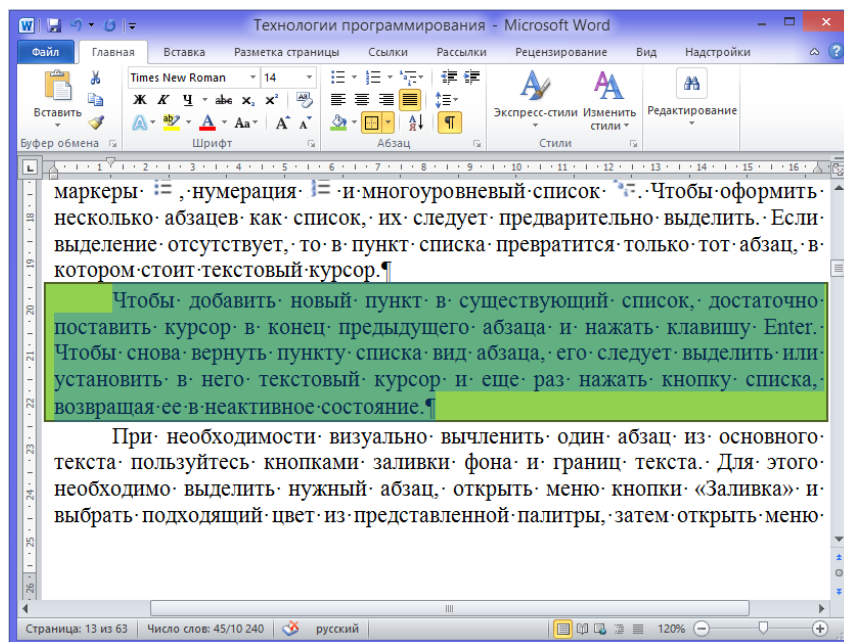


Рис. 1.20. Визуально выделенный абзац

## 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».

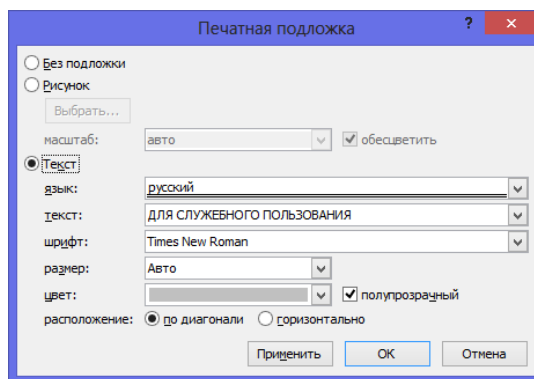


Рис. 1.21. Диалоговое окно «Печатная подложка»

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».



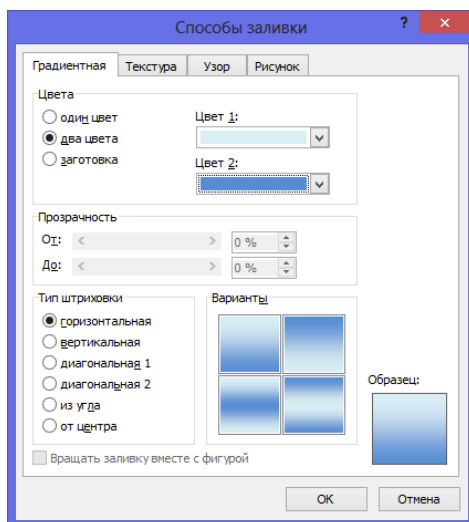


Рис. 1.22. Диалоговое окно «Способы заливки»

При необходимости можно использовать рамки «Границы страницы».

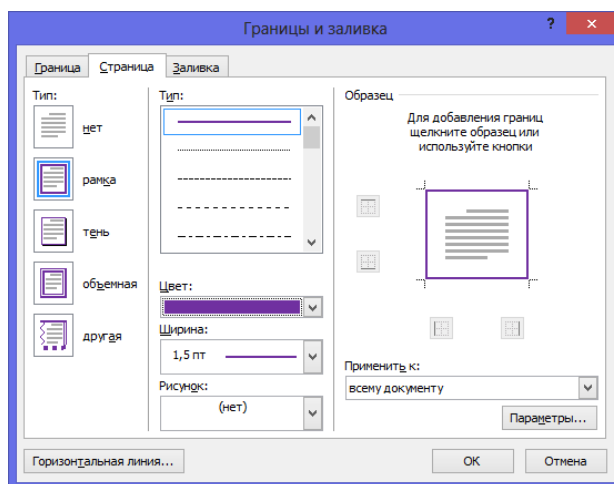


Рис. 1.23. Диалоговое окно «Границы и заливка»

### 1.11 Вставка символов и формул

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

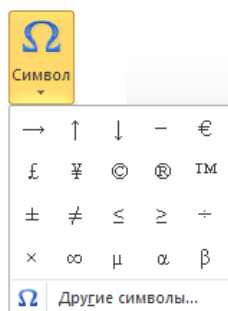


Рис. 1.24. Меню «Символ»

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».

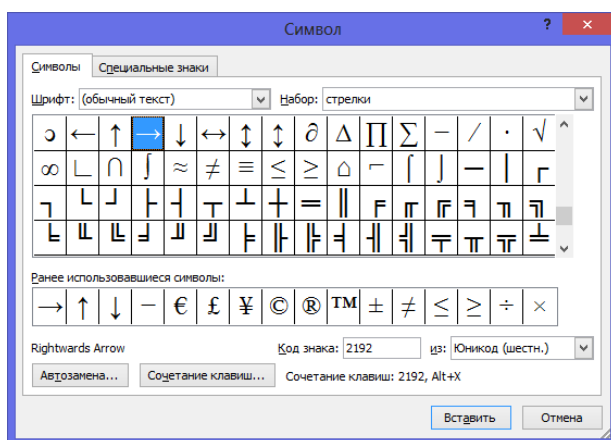


Рис. 1.25. Диалоговое окно «Символ»

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.

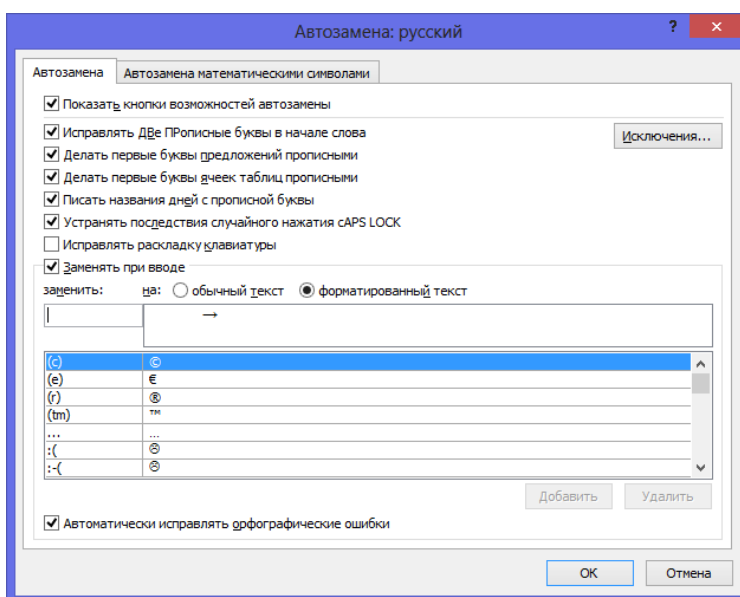


Рис. 1.26. Диалоговое окно «Автозамена»

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.

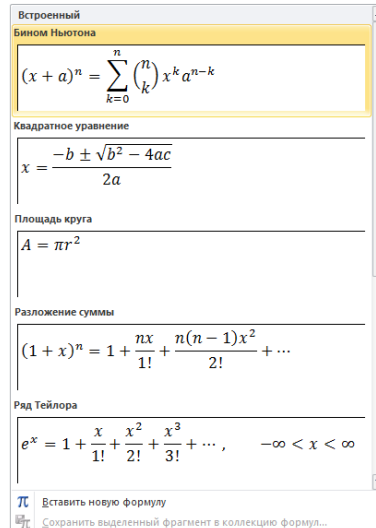


Рис. 1.27. Создание формул

### Задание

1. Открыть файл «Тенденции на рынке труда.doc» в MS Word 2010.
2. В данном документе необходимо сделать:
  1. Выравнивание по ширине;
  2. 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».
  3. Присвоить элементам текста разного уровня соответствующие стили. Основной заголовок – «Заголовок 1», подзаголовок – «Заголовок 2», основной текст – «Обычный» и т.д.
  4. 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»).
  5. 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.

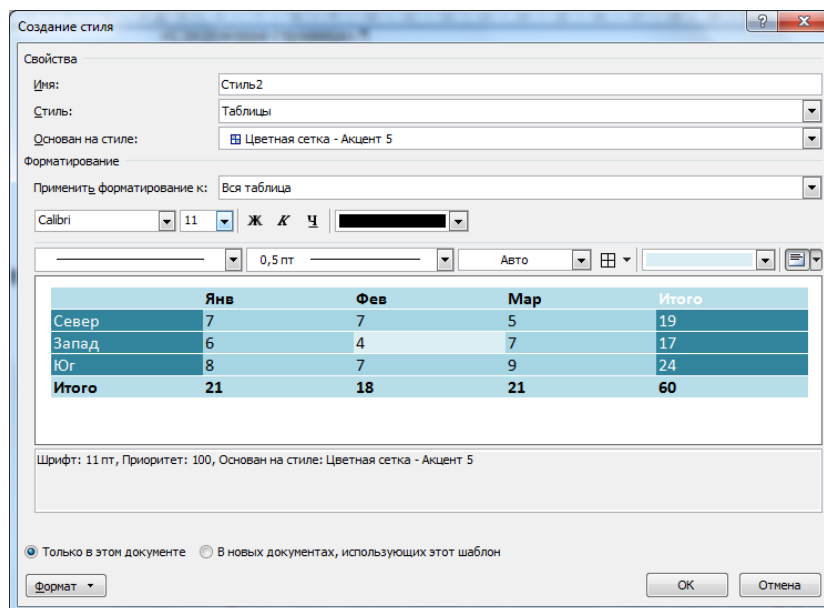


Рис. 1.28. Создание стиля

6. 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, баг-трекинга, ведения wiki проекта и другие.

### **2.1 Создание проекта на BitBucket**

1. Зайдите на сайт <https://bitbucket.org>. Авторизируйтесь под своим аккаунтом google, либо создайте новый. 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 выбрать название надо определиться 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 «Actions» 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 репозиторий с помощью пункта «Downloads».

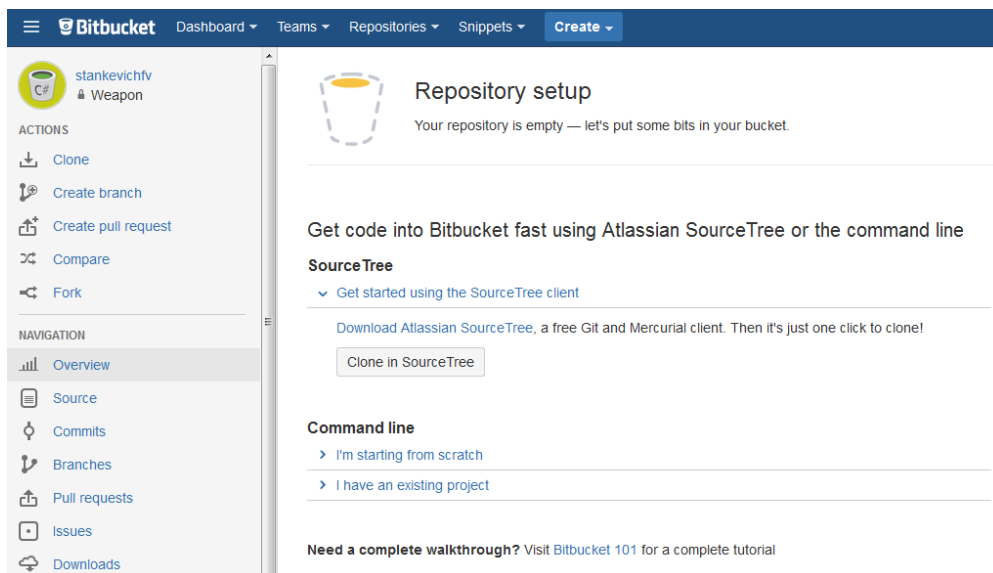


Рис. 2.1 Главная страница BitBucket

## Create a new repository

Owner: stankevichfv

Name: Weapon

Description:

Access level:  This is a private repository

Forking: Allow only private forks

Repository type:  Git,  Mercurial

Project management:  Issue tracking,  Wiki

Language: C#

Repository integrations: HipChat  Enable HipChat notifications

Рис. 2.2. Создание репозитория

## 2.2 Система контроля версий Git

BitBucket – это централизованная система управления версиями. В настоящее время она используется многими сообществами разработчиков открытого программного обеспечения.

Git – распределённая система, где исходный код может храниться в разных хранилищах. Хранилище может располагаться на локальном диске или на сетевом сервере.

Для совместной работы над файлами в Git используется модель копирование – изменение – слияние. Кроме того, для файлов, не допускающих слияние (различные бинарные форматы файлов), можно использовать модель блокирование – изменение – разблокирование.

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).

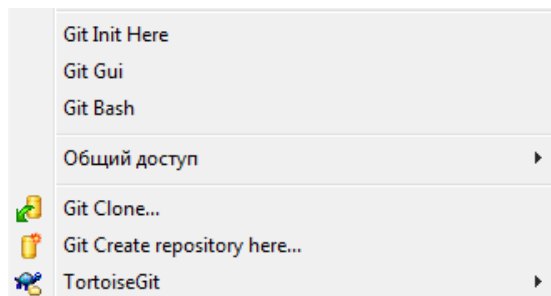


Рис. 2.4. Контекстное меню папки

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 «ОК», начнется скачивание репозитория с Git-сервера.
5. Затем система попросит 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 новые файлы в папку проекта (файл SampleProgram.cs).
2. Нажмите правой кнопкой мыши по папке проекта, из контекстного меню выберите «Git Commit -> “master”...».



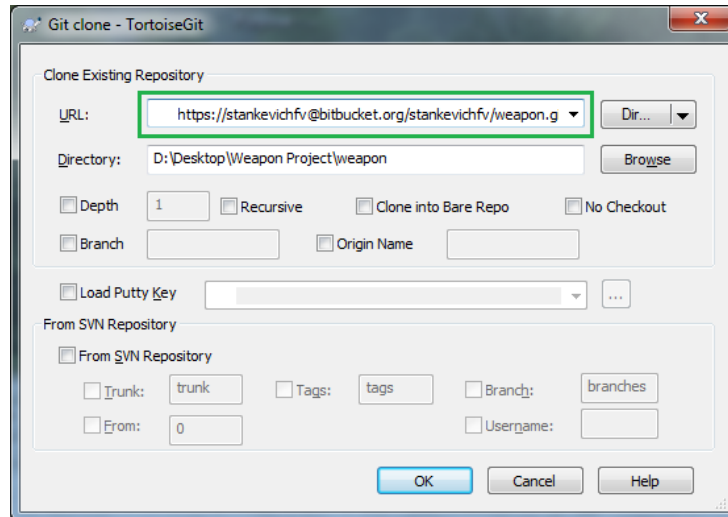


Рис. 2.5. Диалоговое окно «Git clone»

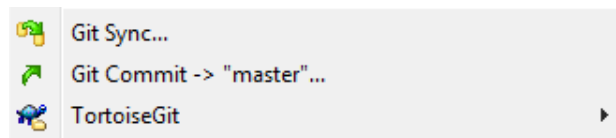


Рис. 2.6. Контекстное меню

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

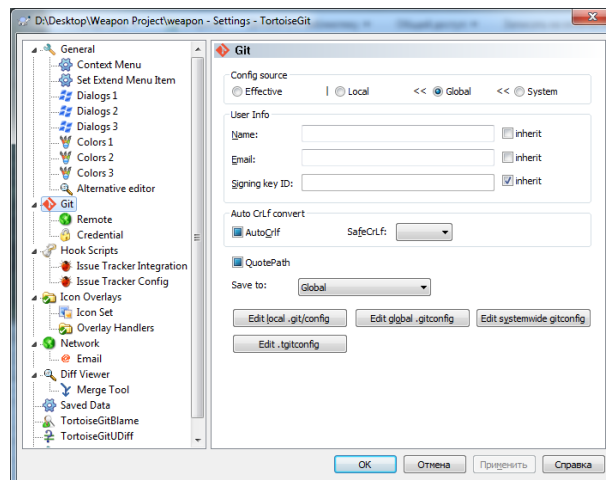


Рис. 2.7. Ввод имени и e-mail

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 изменённых, добавленных и удалённых вами файлов локальной рабочей копии. У изменённых файлов будут автоматически стоять флажки, у остальных — 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 (предназначена 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.

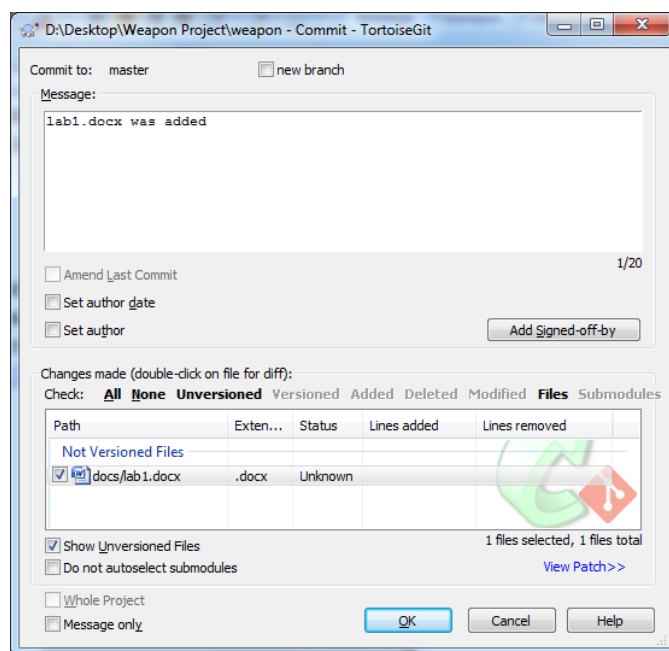


Рис. 2.8. Диалоговое окно «Commit».

6. После выделения нужных файлов или всех сразу, нажмите кнопку «ОК». 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. 2.9).

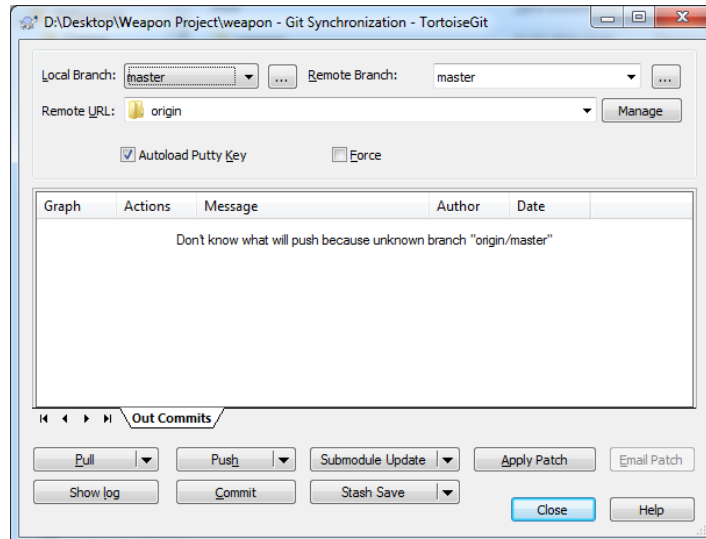


Рис. 2.9. Диалоговое окно «Git Sync».

8. В появившемся окне (рис. 2.9) нажмите «Push» и введите пароль. Кнопка

### Просмотр изменений

Откройте добавленный ранее файл (SampleProgram.cs) в любом текстовом редакторе и внесите в него любое 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).

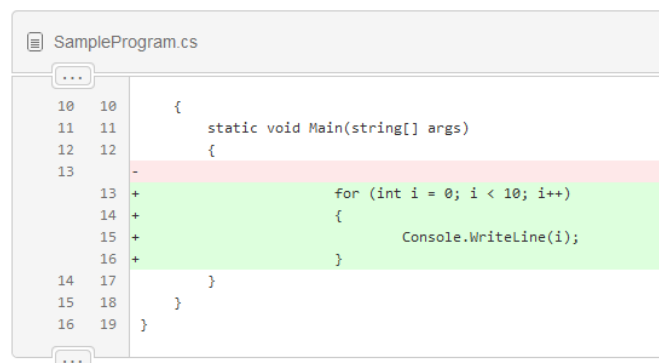


Рис. 2.10. Просмотр изменений

Вызовите 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 детали сообщения разработчиков.

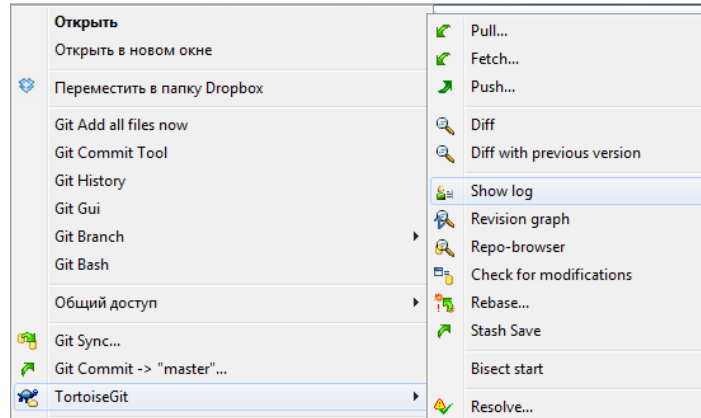


Рис. 2.11. Контекстное меню

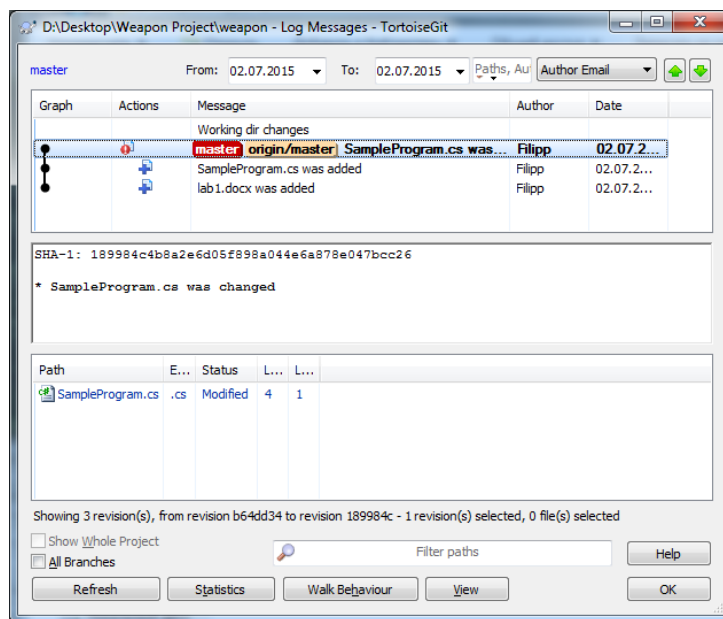


Рис. 2.12. Журнал изменений

Для просмотра отдельных изменений в коде щелкните два раза на изменении из списка. В результате откроется окно утилиты Tortoise Merge, отображающее внесенные изменения (рис. 2.12).

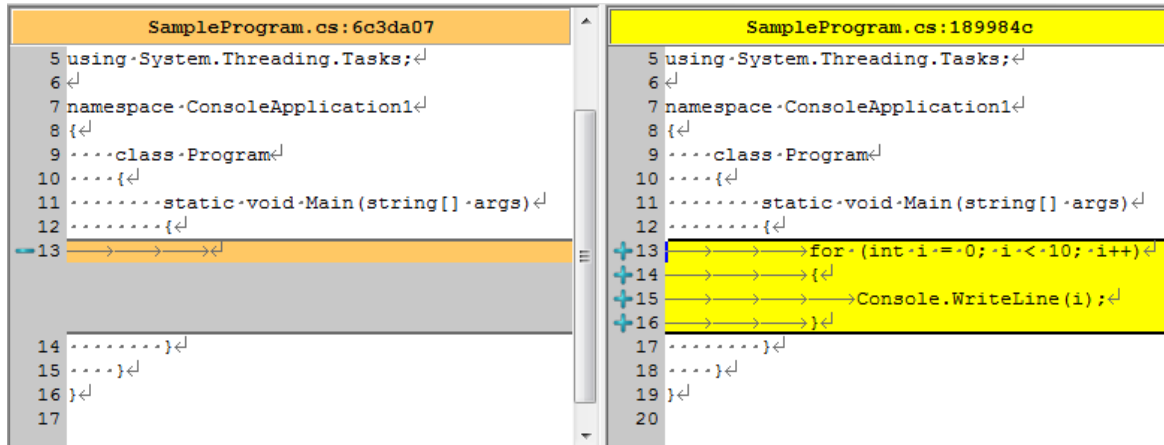


Рис. 2.13. Внесенные изменения

### 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.

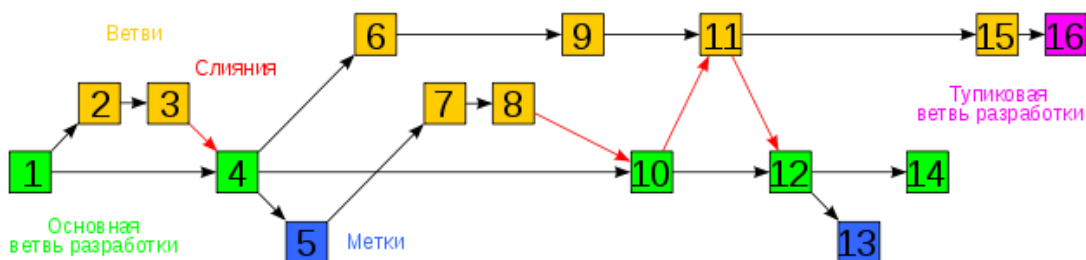


Рис. 2.6. Пример эволюции ветвей

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 именуется *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 осмыслены. 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) – язык графического описания для объектного моделирования в области разработки программного обеспечения. UML является языком широкого профиля, это – открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода.

Использование UML не ограничивается моделированием программного обеспечения. Его также используют для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

### 3.1 Сущности в UML

Объект – реальная именованная сущность, обладающая свойствами и проявляющая свое поведение.

Класс содержит описание данных и операций над ними. В классе дается обобщенное описание некоторого набора родственных, реально существующих объектов. Объект – конкретный экземпляр класса.

В качестве примера можно привести чертеж танка или его описание (класс) и реальный танк (экземпляр класса, или объект) (рис. 43, 44).

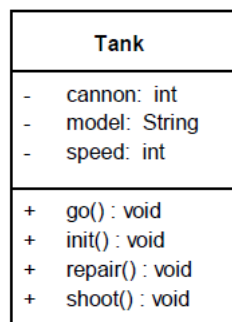


Рис. 3.1. Графическое изображение класса

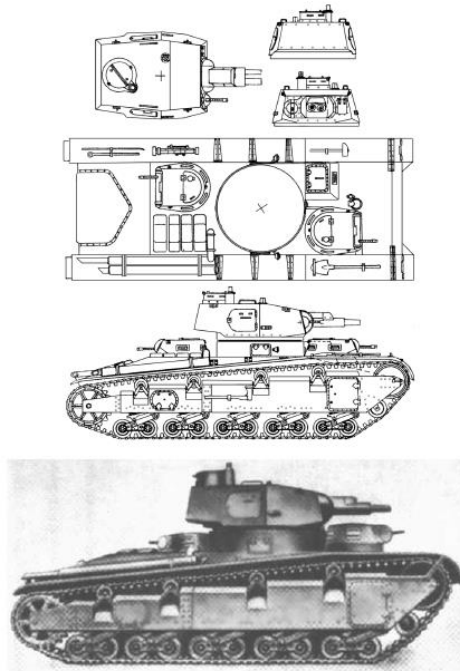


Рис. 3.2. Описание класса (чертеж) и реальный объект

Объектно-ориентированное программирование основано на принципах:

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

Инкапсуляция (*encapsulation*) – принцип, объединяющий данные и код, манипулирующий этими данными, а также защищающий в первую очередь данные от прямого внешнего доступа и неправильного использования. Другими словами, доступ к данным класса возможен только посредством методов этого же класса.

Наследование (*inheritance*) – это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним свойства и методы, характерные только для него.

Наследование бывает двух видов:

- одиночное — класс (он же подкласс) имеет один и только один суперкласс (предок);
- множественное — класс может иметь любое количество предков (в Java запрещено).

Графически наследование часто изображается в виде диаграмм UML (рис. 45).



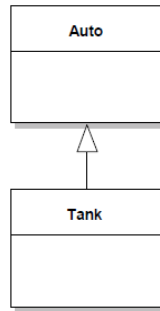


Рис. 3.3. Графическое изображение наследования

Класс «Auto» называется суперклассом, а «Tank» – подклассом.

Полиморфизм (polymorphism) – механизм, использующий одно и то же имя метода для решения двух или более похожих, но несколько отличающихся задач.

Целью полиморфизма применительно к ООП является использование одного имени для задания общих для класса действий. В более общем смысле концепцией полиморфизма является идея «один интерфейс, множество методов».

Механизм «позднего связывания» в процессе выполнения программы определяет принадлежность объекта конкретному классу и производит вызов метода, относящегося к классу, объект которого был использован.

Краеугольным камнем наследования и полиморфизма предстает следующая парадигма: *«объект подкласса может использоваться всюду, где используется объект суперкласса»*.

При вызове метода класса он ищется в самом классе. Если метод существует, то он вызывается. Если же метод в текущем классе отсутствует, то обращение происходит к родительскому классу и вызываемый метод ищется у него. Если поиск неудачен, то он продолжается вверх по иерархическому дереву вплоть до корня (верхнего класса) иерархии [4].

**Класс** имеет название, атрибуты и операции. Класс на диаграмме показывается в виде прямоугольника, разделенного на 3 области. В верхней содержится название класса, в средней – описание атрибутов (свойств), в нижней – названия операций – услуг, предоставляемых объектами этого класса (рис. 46).



Рис. 3.4. Изображение класса в нотации UML

**Атрибуты** класса определяют состав и структуру данных, хранимых в объектах этого класса. Каждый атрибут имеет имя и тип, определяющий, какие данные он представляет. При реализации объекта в программном коде для атрибутов будет выделена память, необходимая для хранения всех атрибутов, и каждый атрибут будет иметь конкретное значение в любой момент времени работы программы. Объектов одного класса в программе может быть сколько угодно много, все они имеют одинаковый набор атрибутов, описанный в классе, но значения

атрибутов у каждого объекта свои и могут изменяться в ходе выполнения программы.

Для каждого атрибута класса можно задать видимость (*visibility*). Эта характеристика показывает, доступен ли атрибут для других классов. В UML определены следующие уровни видимости атрибутов:

- Открытый (*public*) – атрибут виден для любого другого класса (объекта);
- Защищенный (*protected*) – атрибут виден для потомков данного класса;
- Закрытый (*private*) – атрибут не виден внешними классами (объектами) и может использоваться только объектом, его содержащим.

Последнее значение позволяет реализовать свойство инкапсуляции данных. Например, объявив все атрибуты класса закрытыми, можно полностью скрыть от внешнего мира его данные, гарантируя отсутствие несанкционированного доступа к ним. Это позволяет сократить число ошибок в программе. При этом любые изменения в составе атрибутов класса никак не скажутся на остальной части ПС.

Класс содержит объявления операций, представляющих собой определения запросов, которые должны выполнять объекты данного класса. Каждая операция имеет сигнатуру, содержащую имя операции, тип возвращаемого значения и список параметров, который может быть пустым. Реализация операции в виде процедуры – это метод, принадлежащий классу. Для операций, как и для атрибутов класса, определено понятие «видимость». Закрытые операции являются внутренними для объектов класса и недоступны из других объектов. Остальные образуют интерфейсную часть класса и являются средством интеграции класса в ПС.

Надо избегать слишком больших или, наоборот, чересчур маленьких классов. Каждый класс должен хорошо делать что-то одно. Если ваши абстрактные классы слишком велики, то модель будет трудно модифицировать и повторно использовать. Если же они слишком малы, то придется иметь дело с таким большим количеством абстракций, что ни понять, ни управлять ими будет невозможно.

### 3.2 Отношения между сущностями

Взаимосвязь — это особый тип логических отношений между сущностями, показанных на диаграммах классов и объектов. В UML представлены следующие виды отношений:

#### Ассоциация

Ассоциация показывает, что объекты одной сущности (класса) связаны с объектами другой сущности.

Каждая **ассоциация** несет информацию о связях между объектами. Наиболее часто используются бинарные ассоциации, связывающие два класса. Ассоциация может иметь название, которое должно выражать суть отображаемой связи. Помимо названия, ассоциация может иметь такую характеристику, как **множественность**. Она показывает, сколько объектов каждого класса может участвовать в ассоциации. Множественность указывается у каждого конца ассоциации (полюса) и задается конкретным числом или диапазоном чисел. Множественность, указанная в виде звездочки, предполагает любое количество (в том числе, и ноль). Например, ассоциация связывает один объект класса «Набор товаров» с одним или более объектами класса «товар». 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.

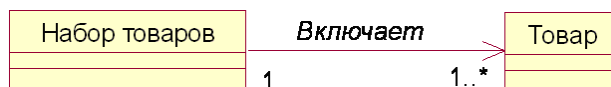


Рис.3.5. Применение ассоциаций

Существует пять различных типов ассоциации. Наиболее распространёнными являются двунаправленная и однонаправленная. Например, классы «рейс» и «самолёт» связаны двунаправленной ассоциацией, а классы «человек» и «кофейный автомат» связаны однонаправленной.

Двойные ассоциации (с двумя концами) представляются линией, соединяющей два классовых блока. Ассоциации более высокой степени имеют более двух концов и представляются линиями, один конец которых идет к классовому блоку, а другой к общему ромбику. В представлении однонаправленной ассоциации добавляется стрелка, указывающая на направление ассоциации.

Ассоциация может быть именованной, и тогда на концах представляющей её линии будут подписаны роли, принадлежности, индикаторы, мультипликаторы, видимости или другие свойства [5].

### Агрегация

Агрегация — это разновидность ассоциации при отношении между целым и его частями. Как тип ассоциации агрегация может быть именованной. Одно отношение агрегации не может включать более двух классов (контейнер и содержимое).

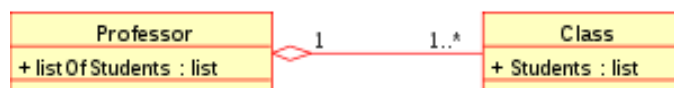


Рис.3.6. Диаграмма классов, показывающая агрегацию между двумя классами

Агрегация встречается, когда один класс является коллекцией или контейнером других. Причём по умолчанию, агрегацией называют *агрегацию по ссылке*, то есть когда время существования содержащихся классов не зависит от времени существования содержащего их класса. Если контейнер будет уничтожен, то его содержимое — нет.

Графически агрегация представляется пустым ромбиком на блоке класса и линией, идущей от этого ромбика к содержащемуся классу.

### Композиция

Композиция — более строгий вариант агрегации. Известна также как агрегация по значению.

Композиция имеет жёсткую зависимость времени существования экземпляров класса контейнера и экземпляров содержащихся классов. Если контейнер будет уничтожен, то всё его содержимое будет также уничтожено.

Графически представляется как и агрегация, но с закрашенным ромбиком.

## Различия между композицией и агрегацией

Целое композиции должно иметь мультипликатор 0..1 или 1, что показывает, что часть является частью только одного целого. В агрегации же может быть любой мультипликатор.

Приведём наглядный пример. Комната является частью квартиры, следовательно здесь подходит композиция, потому что комната без квартиры существовать не может. А, например, мебель не является неотъемлемой частью квартиры, но в то же время, квартира содержит мебель, поэтому следует использовать агрегацию.

## Обобщение (наследование)

Обобщение на диаграммах классов используется, чтобы показать связь между классом-родителем и классом-потомком. Оно вводится на диаграмму, когда возникает разновидность какого-либо класса, а также в тех случаях, когда в системе обнаруживаются несколько классов, обладающих сходным поведением (в этом случае общие элементы поведения выносятся на более высокий уровень, образуя класс-родитель).

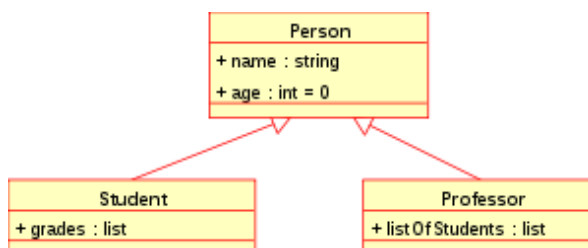


Рис. 3.7. Диаграмма классов, показывающая обобщение

Например: животные — супертип млекопитающих, которые, в свою очередь, — супертип приматов, и так далее. Эта взаимосвязь легче всего описывается фразой «А — это Б» (приматы — это млекопитающие, млекопитающие — это животные).

Графически обобщение представляется линией с пустым треугольником у супертипа.

Обобщение также известно как наследование или «is a» взаимосвязь.

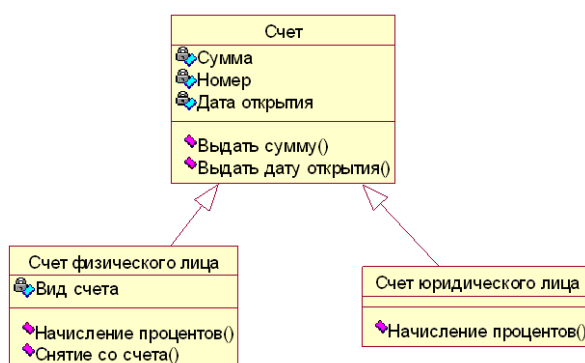


Рис.3.8. Обобщение

UML позволяет строить модели с различным уровнем детализации. На рис.3.9 показана детализация модели, представленной на рис.3.7.

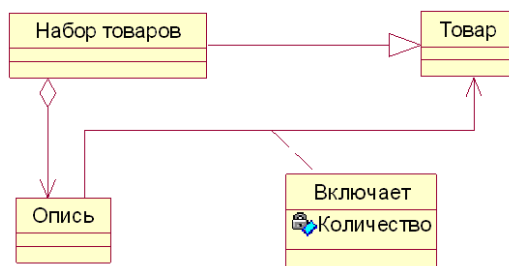


Рис. 3.9. Детализация модели набора товаров

Обобщение показывает, что набор товаров – это тоже товар, который может быть предметом заказа, продажи, поставки и т.д. Набор включает опись, в которой указывается, какие товары входят в набор, а класс-ассоциация «Включает» определяет количество каждого вида товаров в наборе [5].

### Реализация

Реализация — отношение между двумя элементами модели, в котором один элемент (*клиент*) реализует поведение, заданное другим (*поставщиком*). Графически реализация представляется также как и наследование, но с пунктирной линией.

### Зависимость

Зависимость — это слабая форма отношения использования, при котором изменение в спецификации одного влечёт за собой изменение другого, причем обратное не обязательно. Возникает когда объект выступает например в форме параметра или локальной переменной.

Графически представляется пунктирной стрелкой, идущей от зависимого элемента к тому, от которого он зависит. Зависимость может быть между экземплярами, классами или экземпляром и классом.

### Уточнения отношений

Уточнение имеет отношение к уровню детализации. Один пакет уточняет другой, если в нем содержатся те же самые элементы, но в более подробном представлении. Например, при написании книги вы наверняка начнете с формулировки предложения, в котором кратко будет представлено содержание каждой главы. Предположим, что резюме к каждой главе в качестве отдельного элемента входит в пакет «Предложение». Допустим также, что «Завершённая книга» — это пакет, элементами которого являются законченные главы. В этом контексте пакет «Завершённая книга» является уточнением пакета «Предложение».

### Мощность отношений (Кратность)


Мощность отношения (мультипликатор) означает число связей между каждым экземпляром класса (объектом) в начале линии с экземпляром класса в ее конце. В таблице 1 указаны отношения и их объяснение.

## Отношения между сущностями

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

### 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.

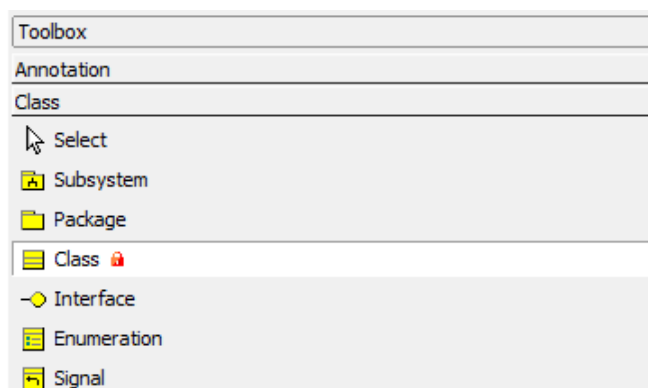


Рис.3.10. Палитра элементов

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).

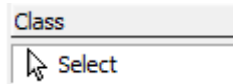


Рис.3.11. Режим редактирования элементов

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).

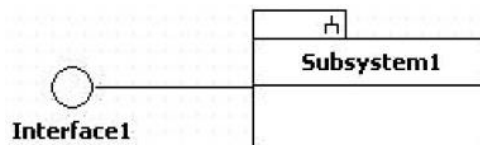


Рис.3.12. Отношение между подсистемой и интерфейсом

### Добавление операции к подсистеме

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

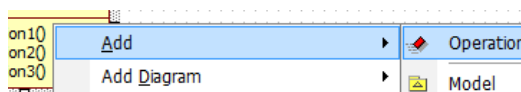


Рис. 3.13. Создание операции

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

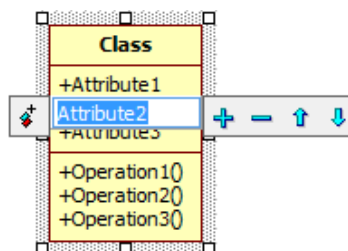


Рис. 3.14. Создание операций и атрибутов

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

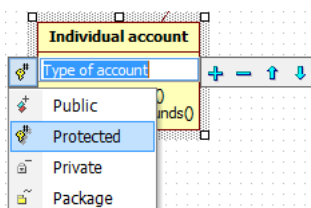


Рис. 3.15. Изменение свойства видимости атрибута

### 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.

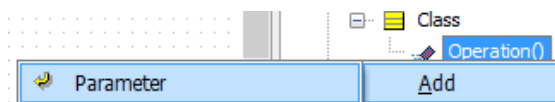


Рис.3.16. Создание параметров

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

## 3.4 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



### 3.5 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 кодировании позволяет разработчикам более эффективно читать и сопровождать программный код (вносить в него изменения). Соглашение о кодировании позволяет сосредоточиться не на том, *как* написан программный код, а на том, *что* в нем написано. В отсутствие оно, разработчику пришлось бы сначала привыкнуть к стилю автора данного кода (разработка на крупных проектах ведется группами), и только потом он сможет уже сосредоточиться на понимании кода. Соглашение о кодировании принимается, как правило, для программного проекта, и позволяет 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 переменным, 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. 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 несут осмысленный характер. Нельзя: *form1*, можно: *ak47*;
11. Имена 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. 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. 3.5).

### 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:

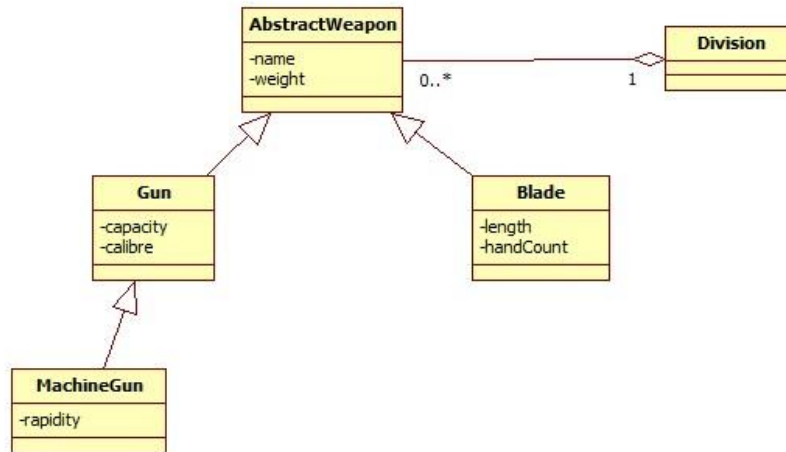


Рис.3.25. Пример диаграммы классов предметной области

### 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. Джим Арлоу, Айла Нейштадт UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, Символ-Плюс, 2007
2. Дж. Рамбо, М. Блаха UML 2.0. Объектно-ориентированное моделирование и разработка, Питер, 2007
3. StarUML. Руководство пользователя [Электронный ресурс]. — Режим доступа: [http://staruml.sourceforge.net/docs/user-guide\(ru\)/user-guide.pdf](http://staruml.sourceforge.net/docs/user-guide(ru)/user-guide.pdf), свободный.
4. Блинов И.Н, Романчик В.С. Java 2. Практическое руководство. — Мн.: УниверсалПресс, 2005. — 400 с.
5. Диаграммы классов UML. Логическое моделирование [Электронный ресурс]. — Режим доступа: <http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=3>, свободный.

## 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 разрабатываться 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.

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.

## 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 согласованы 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).

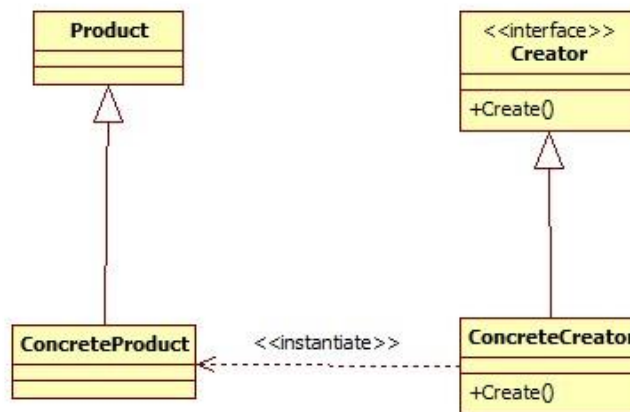


Рис. 4.1. UML диаграмма шаблона Factory Method

### 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 его последующей модификации и читабельности. В таких случаях считается допустимым написать рабочий код, с которыми архитектурами (и не только) нарушениями. Например, вы можете написать логику

расчета стоимости в классе, отвечающем за пользовательский интерфейс (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 = "ТТ";
        gun.Price = 10000;
        gun.Calibre = 5.45;
        gun.Capacity = 8;

        Blade blade = new Blade();
        blade.Name = "Штык-нож";
        blade.Price = 1500;
        blade.Length = 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 = "ТТ";
        gun.Price = 10000;
        gun.Calibre = 5.45;
        gun.Capacity = 8;

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



```

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

        return division;
    }
}

```

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 = "ТТ";
        gun.Price = 10000;
        gun.Calibre = 5.45;
        gun.Capacity = 8;

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

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

        return division;
    }
}

```

Данный пример наглядно демонстрирует последовательное применение рефакторинга, а также использования шаблона Factory Method.

## 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 соответствии 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. 4.1).

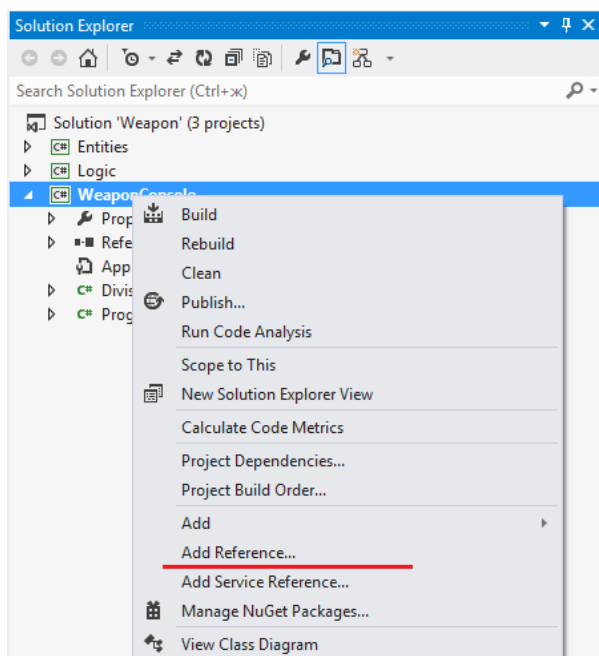


Рис. 4.1 Добавление ссылки на проект

Затем выбрать проекты, на которые необходимо добавить ссылку (те в которых содержатся необходимые классы) (рис. 4.2).

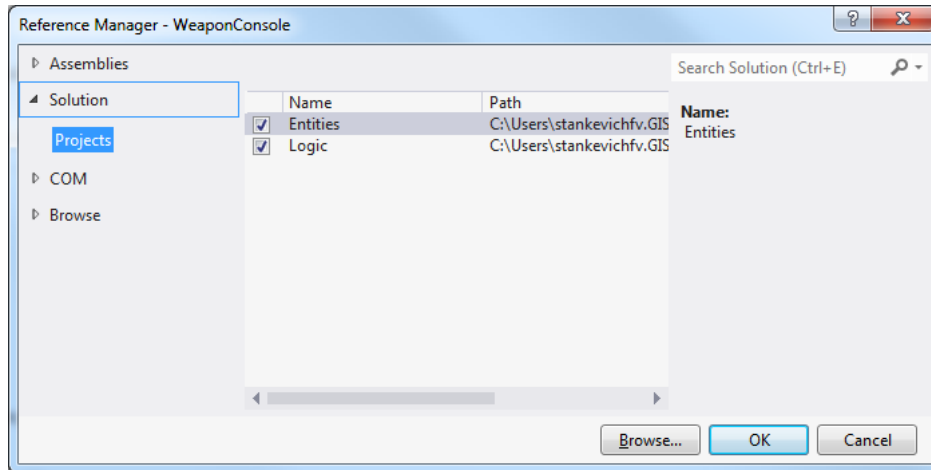


Рис. 4.2 Выбор проектов

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

**Пример:**

```
public class ProductPrinter
{
    public void Print(Product product)
    {
        //вывод данных экран
        Console.WriteLine("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).

4. В приложении необходимо реализовать логирование (информационно-отладочный вывод) посредством реализации интерфейса ILogger. Должно быть, как минимум две реализации, например, логирование 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)
    {
        ...
    }
}
```

### **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.
2. Блинов И.М., Романчик В.С. Java. Промышленное программирование: практ. пособие — Минск: УниверсалПресс, 2007. — 2007.
3. Блинов И.Н., Романчик В.С. Java. Методы программирования, Четыре четверти, 2013
4. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб: Питер, 2007. — 266 с.

## Лабораторная работа № 5 Знакомство с WinForms. Создание графического интерфейса

### Цель работы:

Ознакомиться с WinForms. Создать графический интерфейс для написанного ранее консольного приложения в соответствии с вариантом.

### Введение

В данной лабораторной работе проведем знакомство с элементами среды разработки Visual C#, экспресс-выпуск на примере использования Windows Forms для построения программы C# и создания простого пользовательского интерфейса. Windows Forms предоставляет для проекта такие компоненты, как диалоговые окна, меню, кнопки и многие другие элементы управления, являющиеся частью стандартного пользовательского интерфейса Windows. По существу, эти элементы управления являются просто классами из библиотеки .NET Framework. Представление Конструктор в Visual C#, экспресс-выпуск позволяет перетаскивать элементы управления в основную форму приложения и изменять их размеры и расположение. После этого IDE автоматически добавит исходный код для создания и инициализации экземпляра соответствующего класса.

В создании пользовательских интерфейсов для приложений Windows Forms имеются три основных этапа:

- создание приложения Windows Forms;
- добавление элементов управления на поверхность разработки;
- установка начальных свойств для элементов управления;
- написание обработчиков для заданных событий.

### 5.1 Создание приложения Windows Forms

Откроем Visual Studio и создадим приложение: **Файл** → **Создать** → **Проект**. В появившемся окне во вкладке Visual C# (слева) выберем приложение Windows Forms (рис. 5.1).

Ниже в окне вводим имя проекта и выбираем его расположение.

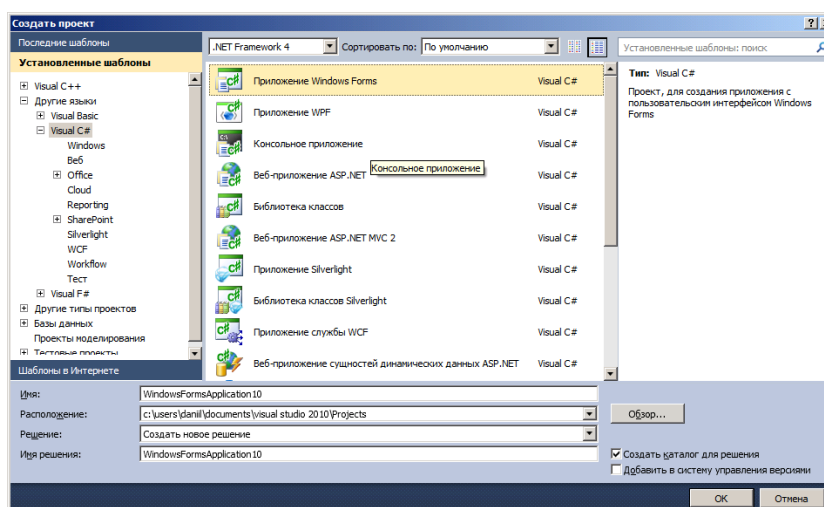


Рис. 5.1. Тип проекта

## 5.2 Добавление элементов управления

Элементы управления, такие как кнопки и текстовые поля, можно перетащить мышью на поверхность разработки, представляющую форму. На рисунке ниже показано поле со списком, которое при помощи перетаскивания из панели элементов было добавлено в форму в конструкторе Windows Forms.

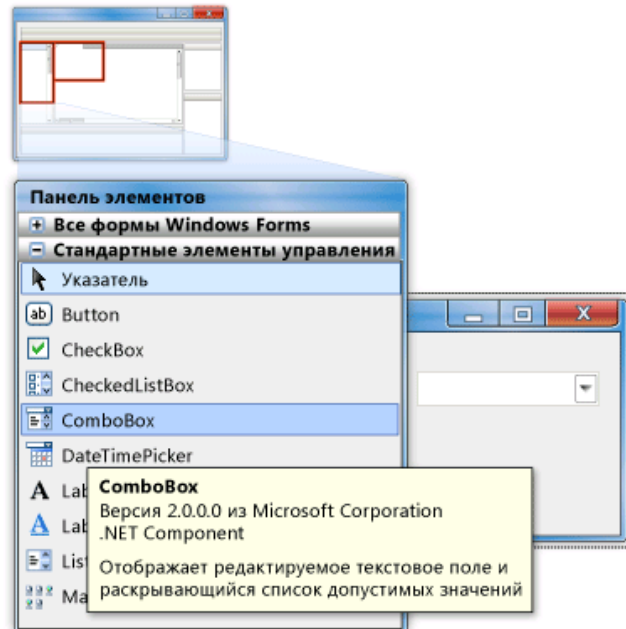


Рис. 5.2. Добавление элемента управления

## 5.3 Задание свойств

После добавления элемента управления в форму в окне Свойства можно задать его свойства, такие как цвет фона и текст по умолчанию. Значения, задаваемые в окне Свойства, являются начальными значениями, которые будут назначены соответствующему свойству при создании элемента управления во время выполнения. Во многих случаях доступ к значениям и их изменение возможно программными средствами во время выполнения путем получения или установки свойств в экземпляре класса элемента управления в приложении. Окно «Свойства» может оказаться полезным во время выполнения, так как с его помощью можно просматривать все свойства, события и методы, поддерживаемые элементом управления.

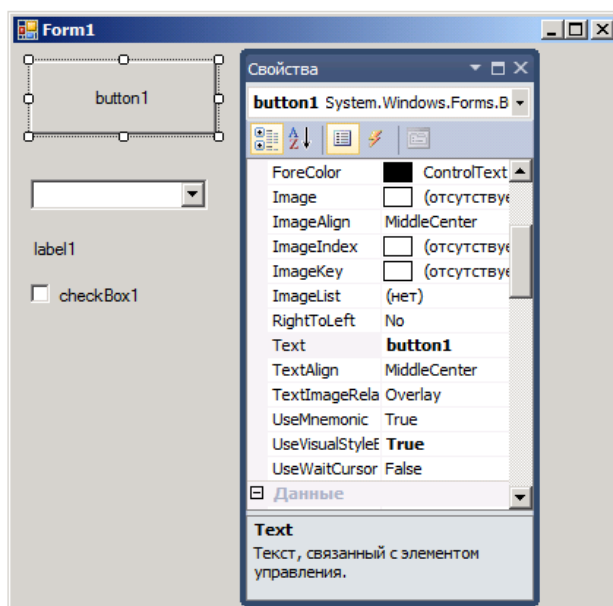


Рис. 5.3. Свойства элемента Button

#### 5.4 Список элементов.

В таблице 3 представлен список элементов Windows forms в C# и их описание.

Таблица 3

*Список элементов Windows Forms*

<b>Элемент</b>	<b>Описание</b>
Button	Элемент управления Windows Forms Button (кнопка) служит для выполнения действия с помощью мыши. На элементе управления Button может отображаться как текст, так и рисунок. Если нажать кнопку, она выглядит так, словно она нажата и отпущена.
RadioButton	Элементы управления Windows Forms RadioButton (переключатели) обеспечивают выбор из двух или более взаимоисключающих вариантов. Функции переключателей и флажков могут показаться схожими, но между ними есть важное отличие: в случае переключателя пользователь может выбрать лишь один вариант.
CheckBox	Элемент управления Windows Forms CheckBox указывает, включено или отключено какое-либо конкретное условие. Обычно он используется для представления пользователю выбора типа "Да/Нет" или "Истина/Ложь". Элементы управления типа "флажок" можно объединять в группы для предоставления пользователю нескольких вариантов выбора
Label	Элементы управления Windows Forms Label предназначены для отображения текста или изображений, которые пользователь не может изменить с клавиатуры. Они используются для идентификации объектов в форме – например, для описания, что произойдет с элементом управления после выполнения на нем щелчка мышью, или для отображения сведений в ответ на процесс



	или событие времени выполнения в приложении.
LinkLabel	Элемент управления форм LinkLabel позволяет добавлять ссылки в стиле веб в приложения форм Windows. Элемент управления LinkLabel может использоваться для тех же целей, что и элемент управления Label; кроме того, имеется возможность задать часть текста в качестве ссылки на объект или веб-страницу.
TextBox	Элемент управления TextBox обычно используется для редактируемого текста, хотя его можно также сделать доступным только для чтения. В текстовых полях можно выводить несколько строк текста, размещать текст в соответствии с размером элемента управления и применять основные элементы форматирования. В элементе управления TextBox можно вводить или отображать текст только в одном формате.
RichTextBox	Элемент управления Windows Forms RichTextBox используется для отображения, ввода и изменения текста с форматированием. Элемент управления RichTextBox выполняет те же функции, что и элемент управления TextBox, но помимо этого он позволяет отображать шрифты, цвета и ссылки, загружать текст и вложенные изображения из файлов, отменять и повторять операции редактирования, а также искать заданные знаки. Элемент управления RichTextBox обычно используется для предоставления возможностей изменения и отображения текста, схожих с возможностями текстовых редакторов, таких как Microsoft Word.
CheckListBox	В элементе управления Windows Forms CheckedListBox отображается список элементов, как и в элементе управления ListBox, а также может отображаться флажок рядом с элементами списка.
ComboBox	Элемент управления Windows Forms ComboBox используется для вывода данных в раскрывающемся поле со списком. По умолчанию элемент управления ComboBox отображается в виде двух частей: верхняя часть представляет собой текстовое поле, в которое пользователь может ввести элемент списка. Вторая часть представляет собой список элементов, один из которых пользователь может выбрать.
ListBox	Элемент управления ListBox в Visual Basic 6.0 заменяется в Visual Basic 2008 либо элементом управления ListBox, либо элементом управления CheckedListBox. Имена некоторых свойств, методов, событий и констант различаются, а в некоторых случаях имеется различие в их поведении.
NumericUpDown	Элемент управления Windows Forms NumericUpDown представляет собой сочетание текстового поля и пары кнопок со стрелками для выбора значения пользователем. Он выводит и задает отдельное числовое значение в списке вариантов. Пользователь может увеличивать и уменьшать число, нажимая кнопки со стрелками вверх и вниз или клавиши со стрелками ВВЕРХ и ВНИЗ, а также вводя число в поле. Одним из вариантов

	применения этого элемента управления является его использование в качестве регулятора громкости музыкального проигрывателя.
MaskedTextBox	Данные можно привязать в элементу управления MaskedTextBox, как к любому другому элементу управления Windows Forms. При этом, если формат данных в базе данных не соответствует формату, ожидаемому определением маски, его необходимо изменить. В следующей процедуре показано, как для этой цели можно использовать события Format и Parse класса Binding для отображения отдельных полей базы данных номеров телефонов и добавочных номеров в качестве одного изменяемого поля.
ToolTip	Элемент управления Tooltip отображает подсказку, которую можно настроить, может содержать изображения, и его можно форматировать.
ProgressBar	Элемент управления форм Windows Forms ProgressBar показывает, на какой стадии находится выполняемое действие, что выражается в соответствующем числе прямоугольников, расположенных на горизонтальной шкале. Завершение действия характеризуется заполненной шкалой. Индикаторы хода работы обычно используются, чтобы показать пользователю время, оставшееся до завершения длительного действия, например при загрузке большого файла.
DatetimePicker	Элемент управления DateTimePicker имеет два состояния. По умолчанию элемент управления DateTimePicker выглядит как текстовое поле с раскрывающимся списком в виде стрелки. Когда пользователь нажимает на стрелку раскрывающегося списка, появляется календарь. При использовании этого элемента управления пользователь может выбрать только одну дату. Элемент управления DateTimePicker также позволяет отображать время вместо дат.
MonthCalendar	Элемент управления MonthCalendar позволяет отображать календарь для одного или нескольких месяцев. При этом пользователи могут выбирать отдельную дату или диапазон дат. В отличие от аналогичного элемента управления DateTimePicker, данный элемент управления позволяет выделить диапазон дат; однако элемент управления DateTimePicker может использоваться для задания не только даты, но и времени.
Timer	Элементы управления timer отвечают на прохождение отрезков времени. Они независимы от пользователя, и их можно программировать на равномерное прохождение действий. Обычно ответ проверяет по системным часам, пришло ли время выполнить задачу. Таймеры полезны и для других видов фоновой обработки.
ListView	В элементе управления Windows Forms ListView отображается список элементов со значками. Представление в виде списка может использоваться для создания пользовательского интерфейса, аналогичного правой области окна Windows Explorer.

PictureBox	Элемент управления Windows Forms PictureBox предназначен для отображения графических объектов в различных форматах. Это может быть растровое изображение (файл BMP), пиктограмма (файл ICO), метафайл (файл WMF или EMF), а также файлы GIF и JPEG.
TreeView	Элемент управления Windows Forms TreeView отображает иерархию узлов аналогично функции отображения файлов и папок в левой области окна проводника Windows.
WebBrowser	Элемент управления WebBrowser предназначен для работы только в среде с полным доверием. Отображаемое в элементе управления HTML-содержимое может поступать от внешних веб-серверов и может содержать неуправляемый код в форме скриптов или веб-элементов управления. Если в такой ситуации используется элемент управления WebBrowser, этот элемент управления является не менее безопасным, чем Internet Explorer, причем управляемый элемент управления WebBrowser не предотвращает запуск такого неуправляемого кода [3].

После ознакомления с элементами форм можно приступить к созданию графического интерфейса для написанного в четвертой лабораторной работе консольного приложения.

## Задание

Необходимо создать графический интерфейс написанного ранее консольного приложения.

1. К созданному в 4 лабораторной работе решению (solution) необходимо добавить проект Windows Forms и назначить его запускаемым проектом (рис. 5.1), вместо проекта консольного приложения.

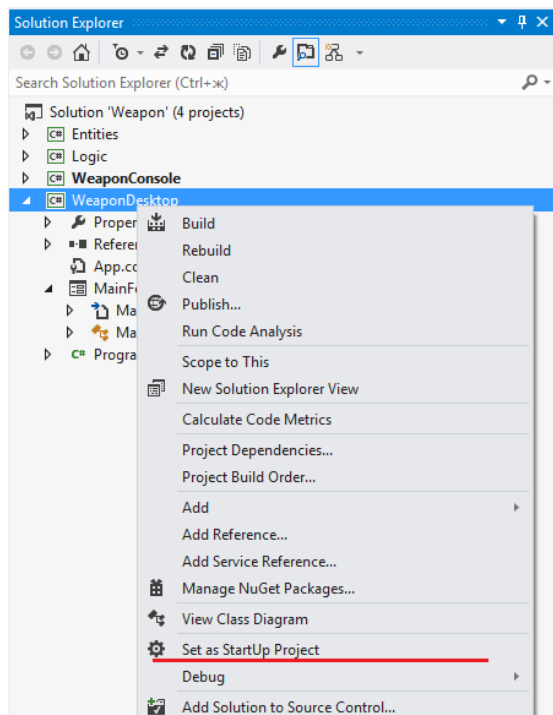


Рис. 5.1 Назначение проектом по умолчанию

2. На главной форме необходимо реализовать следующие элементы:
  - Список созданных объектов (например, с помощью элемента ListBox);
  - Кнопку добавления нового объекта, по нажатию на которую должна появляться новая форма с выбором класса создаваемого объекта и набором полей ввода атрибутов;
  - Кнопку удаления выделенного объекта из списка;
  - Кнопку редактирования выделенного объекта из списка, по нажатию на которую должна появляться новая форма с атрибутами объекта;
  - Текстовый элемент (например, Label) содержащий рассчитанный параметр вашего задания. Этот элемент должен автоматически обновляться при добавлении/удалении/изменении объектов.
3. На форме добавления и изменения объектов должны быть кнопки *ОК* и *Отмена*.

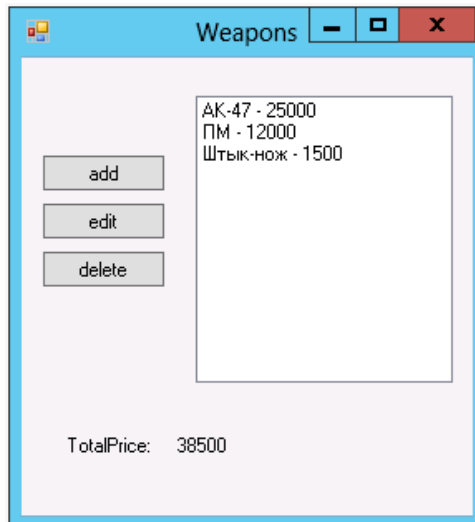


Рис. 5.2 Пример созданного интерфейса

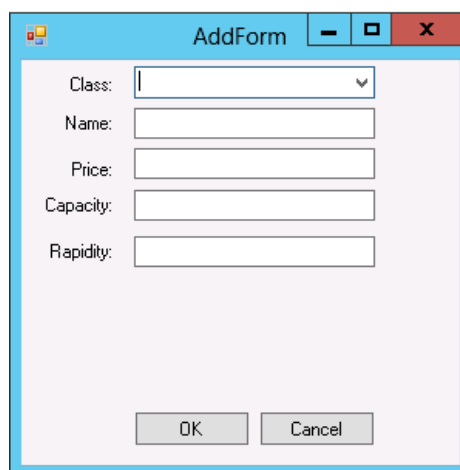


Рис. 5.3 Пример окна добавления объекта

### Требования и рекомендации:

1. Иерархия классов не должна измениться (как в лабораторных работах 3 и 4);
2. Логика приложения и графический интерфейс должны быть отделены друг от друга;
3. Расчет цены (требуемых в задании параметров) должен производиться посредством созданного ранее класса *DivisionCalculator* (Для этого главная форма должна содержать *Division* в качестве поля);
4. Начальный список объектов формы (при ее загрузке) желательно инициализировать с помощью ранее созданной фабрики;
5. Форма добавления объектов может быть использована для редактирования существующего объекта. Для этого необходимо реализовать 2 конструктора формы: один по умолчанию (без параметров) для добавления нового объекта, другой с параметром типа *AbstractEntity*, для редактирования объекта.

### Пример кода:

```
public class AddEditForm
{
    public AddEditForm() //конструктор для добавления
    {
        InitializeComponent();
    }

    //конструктор для редактирования
    public AddForm(AbstractWeapon weapon) : this()
    {
        if (weapon!=null)
        {
            //здесь поля формы заполняются значениями, полученными из Weapon
        }
    }
}
```

6. Для получения готового объекта из формы редактирования необходимо в коде формы редактирования создать новый объект класса *AbstractEntity*. Значения атрибутов взять из полей ввода.

### Пример кода:

```
private AbstractWeapon _weapon;

public AbstractWeapon Weapon
{
    get { return _weapon; }
    set { _weapon = value; }
}

private DialogResult _result;

public DialogResult Result
{
    get { return _result; }
    set { _result = value; }
}

private void okButton_Click(object sender, EventArgs e)
{
    weapon = new Gun(); //new Blade() ....

    //инициализация полей объекта Weapon из полей ввода формы
    weapon.Name = nameTextBox.Text;
    ...
    Result = DialogResult.OK;
}
```

7. Для обработки кнопок ОК и Cancel на форме редактирования можно завести поле типа *DialogResult*:

```
private DialogResult _result;
```

```
public DialogResult Result
{
    get { return _result; }
    set { _result = value; }
}
```

И при нажатии на соответствующую кнопку присваивать ему значения *DialogResult.OK* или *DialogResult.Cancel*:

#### Пример кода:

```
private void CancelButton_Click(object sender, EventArgs e)
{
    Result = DialogResult.Cancel;
}
```

А на главной форме проверять значение поля *Result* для добавления объекта в список:

#### Пример кода:

```
if (AddForm.Result == DialogResult.OK)
{
    AbstractWeapon weapon = AddForm.Weapon;
    this.weaponListBox.Items.Add(weapon);
    division.AddWeapon(weapon);
}
```

8. Количество модальных окон должно быть ограничено (максимально возможно 2 одновременно используемых окна).
9. Необходимо контролировать диапазон допустимых значений для полей ввода.

### Контрольные вопросы

1. Каким образом можно добавить элемент управления на форму?
2. Назовите основные этапы при создании пользовательских интерфейсов для приложений Windows Forms?
3. Назовите все элементы формы, позволяющие отображать текст либо графические объекты?
4. Назовите элемент формы, отвечающий за прохождение отрезков времени

### Список литературы

1. Уотсон К., Нейгел К. Visual C# 2008. Базовый курс. — М.: Изд. дом «Вильямс», 2009. — 1216 с.
2. Шилдт Г. C# 4.0 полное руководство. — М.: Изд. дом «Вильямс», 2011. — 1056 с.
3. Элементы управления для использования в формах Windows Forms [Электронный ресурс]. — Режим доступа: [https://msdn.microsoft.com/ru-ru/library/3xdhey7w\(v=vs.110\).aspx](https://msdn.microsoft.com/ru-ru/library/3xdhey7w(v=vs.110).aspx), свободный.

## Лабораторная работа №6 Issue tracker на примере BitBucket

### Цель работы:

Получить опыт практической работы в паре при создании, разработки ПО, а так же его тестировании и исправлении дефектов.

### **Введение**

Issue tracker – это система для отслеживания и управления заявками (ошибками в ПО, различными вопросами в тех. поддержку и задачами, которые необходимо решить).

Разновидностью Issue tracker систем являются багтрекеры – системы отслеживания/учета ошибок.

Система отслеживания/учета ошибок (англ. *bug tracking system*) – прикладная программа, разработанная с целью помочь разработчикам программного обеспечения (программистам, специалистам по тестированию и др.) учитывать и контролировать ошибки (баги), найденные в программах, пожелания пользователей, а также следить за процессом устранения этих ошибок и выполнения или невыполнения пожеланий.

По сути это программа или веб-приложение, в котором пользователи могут добавлять записи об ошибках (задачах, вопросах), назначать ответственное за устранение ошибки (решение задачи) лицо и статус (открыта заявка, в процессе решения, закрыта или требует каких-либо уточнений). Здесь же по каждому вопросу пользователи могут оставлять комментарии и какую-то дополнительную информацию.

Развитые системы позволяют также автоматизировано вести учет времени, разделять каждую задачу на подзадачи, автоматизировано рассылают уведомления о поступлении заявок/ошибок или комментариев к ним, об изменении статуса и т. д.

Задачам и вопросам можно назначать приоритеты.

При необходимости записи все записи группируются по принадлежности к проекту и его версии (в случае разработки ПО; в случае оказания услуг или тех. поддержки группировка может быть по номеру договора, оказываемой услуге или иному процессу) и могут быть отфильтрованы по этим признакам.

Иногда для каждой задачи указывается стоимость, временные рамки исполнения, зависимости от других задач (какие задачи не могут быть выполнены без выполнения данной задачи или из-за невыполнения каких задач не может быть выполнена данная).

Наиболее сложные и функциональные системы часто имеют возможность взаимодействия с другими системами для получения дополнительных данных или установки связей всех этапов работ (так, например Issue tracking системы, используемые при разработке ПО могут быть связаны с системами контроля версий, репозиториями исходных кодов, базами знаний и т. д.).

### **6.1 Работа в BitBucket**

BitBucket – это сервис для разработчиков программного обеспечения. Один из основных сервисов – система контроля версий. Доступные системы контроля версий – Git и Mercurial. В системе контроля версий есть репозиторий для хранения



исходного кода (пункт меню «Source»), и можно посмотреть любую версию исходного кода. Также можно просмотреть разницу (diff) между двумя последовательными версиями или посмотреть историю изменений отдельного файла (рис. 6.1).

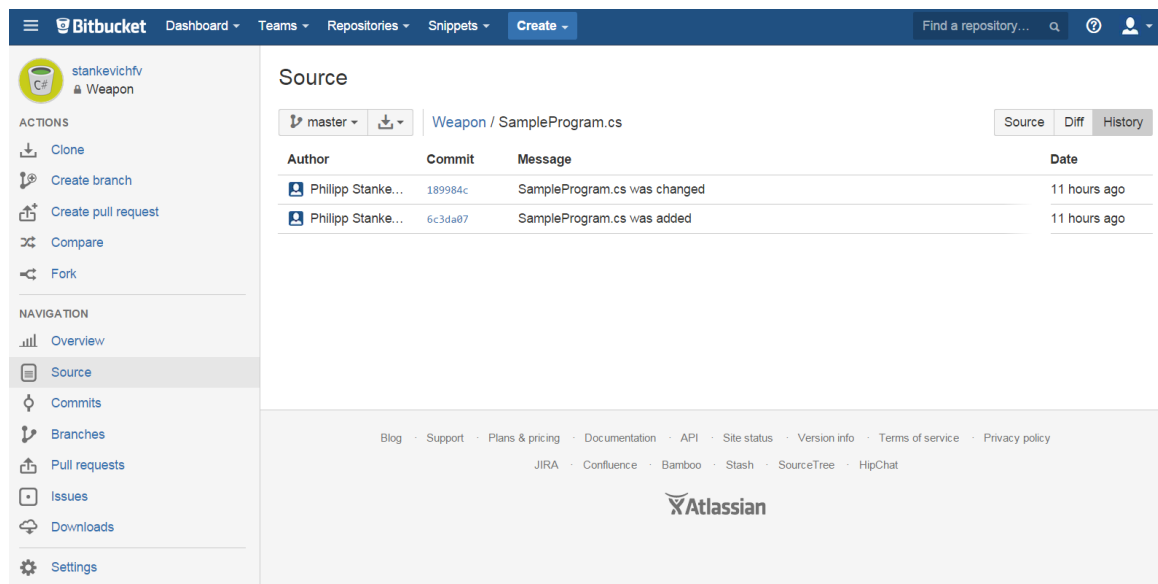


Рис. 6.1. Система контроля версий в BitBucket

Помимо контроля версий BitBucket также предоставляет возможность использования Issue Tracker (пункт меню «Issues»). Если в программе была найдена ошибка, то её можно там зарегистрировать с описанием, менеджер проекта может назначить её исправление определенному разработчику, а тот исправить.

## 6.2 Состав информации о дефекте

Главный компонент такой системы – база данных, содержащая сведения об обнаруженных дефектах. Эти сведения могут включать в себя:

- номер (идентификатор) дефекта;
- кто сообщил о дефекте;
- дата и время, когда был обнаружен дефект;
- версия продукта, в которой обнаружен дефект;
- серьёзность (критичность) дефекта и приоритет решения;
- описание шагов для выявления дефекта (воспроизведения неправильного поведения программы);
- кто ответственен за устранение дефекта;
- обсуждение возможных решений и их последствий;
- текущее состояние (статус) дефекта;
- версия продукта, в которой дефект исправлен.

Кроме того, развитые системы предоставляют возможность прикреплять файлы, помогающие описать проблему (например, дампы памяти или скриншоты).

### 6.3 Жизненный цикл дефекта

Как правило, система отслеживания ошибок использует тот или иной вариант «жизненного цикла» ошибки, стадия которого определяется текущим состоянием, или статусом, в котором находится ошибка.

Типичный жизненный цикл дефекта:

1. Новый – дефект зарегистрирован специалистом по тестированию
2. Назначен – назначен ответственный за исправление дефекта
3. Разрешён – дефект переходит обратно в сферу ответственности тестировщика. Как правило, сопровождается резолюцией, например:
  - Исправлено (исправления включены в версию №...)
  - Дубль (повторяет дефект, уже находящийся в работе)
  - Не исправлено (работает в соответствии со спецификацией, имеет слишком низкий приоритет, исправление отложено до следующей версии и т.п.)
  - «У меня всё работает» (запрос дополнительной информации об условиях, в которых дефект проявляется)
4. Далее специалист по тестированию ПО проводит проверку исправления, в зависимости от чего дефект либо снова переходит в статус *Назначен* (если он описан как исправленный, но не исправлен), либо в статус *Закрит*.
5. Открыт повторно – дефект вновь найден в другой версии.

Система может предоставлять администратору возможность настроить, какие пользователи могут просматривать и редактировать ошибки в зависимости от их состояния, переводить их в другое состояние или удалять.

В корпоративной среде, система отслеживания ошибок может использоваться для получения отчётов, показывающих продуктивность программистов при исправлении ошибок. Однако, часто такой подход не даёт достаточно точных результатов, из-за того что разные ошибки имеют различную степень серьёзности и сложности. При этом серьёзность проблемы не имеет прямого отношения к сложности устранения ошибки.

### 6.4 Примеры систем отслеживания ошибок

Свободно распространяемые	Проприетарные	Разное
<ul style="list-style-type: none"><li>• Redmine;</li><li>• BUGS — the Bug Genie;</li><li>• Bugzilla;</li><li>• eTraxis;</li><li>• GNATS;</li><li>• Mantis bug tracking system;</li><li>• Trac;</li><li>• EmForge;</li><li>• Picket;</li><li>• Flyspray;</li><li>• DEVPROM.</li></ul>	<ul style="list-style-type: none"><li>• Atlassian JIRA;</li><li>• Bontq;</li><li>• PVCS Tracker;</li><li>• Project Kaiser;</li><li>• TrackStudio Enterprise;</li><li>• YouTrack.</li></ul>	<ul style="list-style-type: none"><li>• BugTracker.NET;</li><li>• ClearQuest;</li><li>• Intland CodeBeamer;</li><li>• LifeTask.ru;</li><li>• FlySpray;</li><li>• StarTeam.</li></ul>

## Задание

1. Необходимо протестировать Windows-приложение, написанное в лабораторной работе №5, для этого студентам необходимо разбиться на пары. Один будет выполнять роль *разработчика*, другой — *специалиста по тестированию*.

2. В репозитории необходимо выбрать вашу команду, созданную в лаб. раб. № 2 (рис. 6.2)

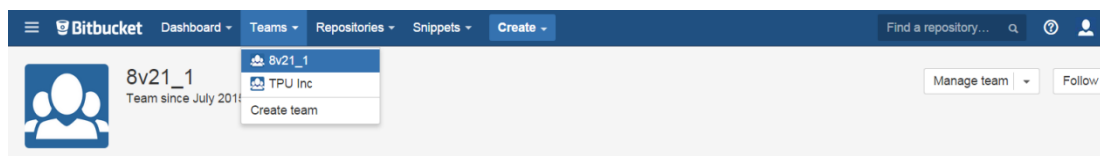


Рис. 6.2. Выбор команды

Если команда не была создана, то необходимо выбрать пункт «Teams» → «Create team» (рис. 6.3).

The screenshot shows the 'Create a team' form. It includes fields for 'Team name' (8v21\_1) and 'Team ID' (8v21\_1), with a green checkmark next to the ID. Below these fields, it states 'Your team will be available at https://bitbucket.org/8v21\_1'. To the right, under 'What is a team?', there is explanatory text and four checked checkboxes: 'Create team-owned repositories', 'Delegate administration', 'Send email invitations', and 'Manage repository access via groups'. Below this is the 'Add team members' section with an input field and an 'Add' button. At the bottom, there is a 'HipChat integration' section with a checkbox for 'Add HipChat for your new team'. The form ends with 'Create' and 'Cancel' buttons.

Рис. 6.3. Создание команды

3. Необходимо нажать кнопку «Manage team» в левом верхнем углу окна (рис. 6.2), затем добавить новую группу «Quality engineers» нажав кнопку «Add Group» и выставить уровень доступа «read» этой группы. Результат показан на рис. 6.4.

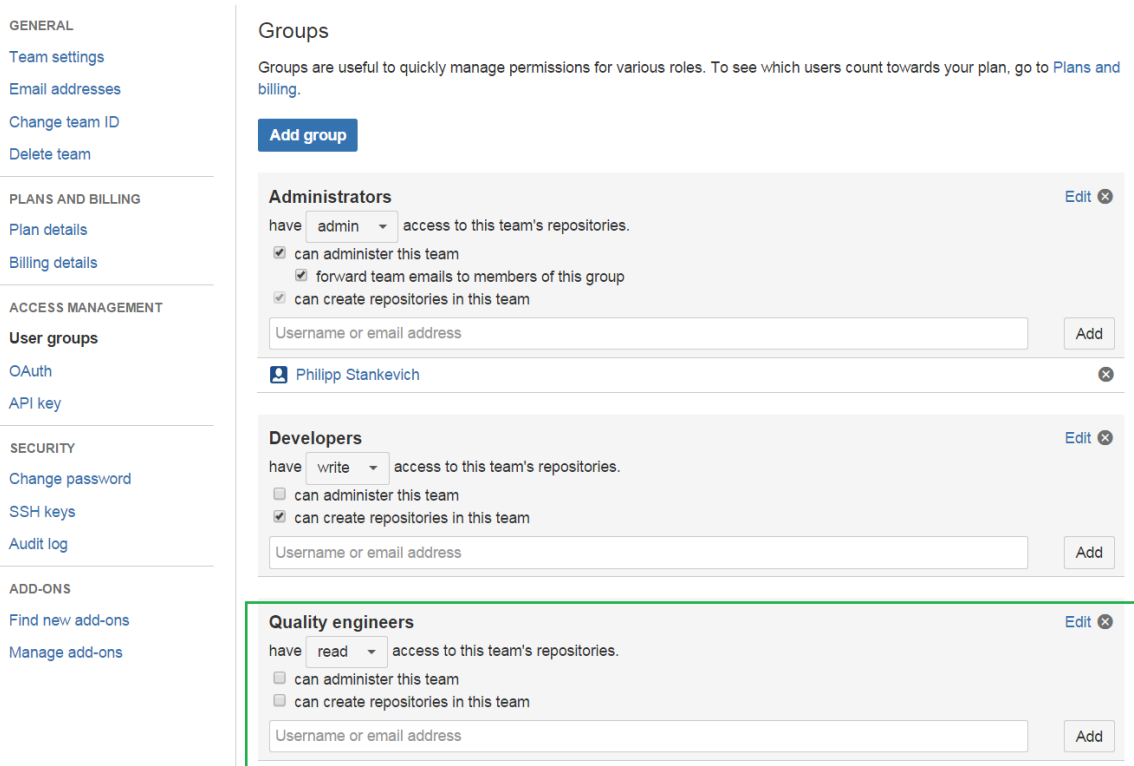


Рис. 6.4. Создание группы и добавление в нее пользователя

4. Затем необходимо ввести имя (или e-mail) пользователя, который будет исполнять роль специалиста по тестированию, в блок группы «Quality engineers» и нажать кнопку «Add».

5. Специалист по тестированию выполняет «Clone» проекта, выполненного в рамках лаб. раб. №5. Инструкцию по выполнению клонирования см. в лаб. раб. №2. Затем специалист по тестированию проверяет программу на наличие ошибок, в качестве которых может служить заполнение поля ввода отрицательными или некорректными значениями. Далее специалист по тестированию создает *issue* типа *bug* в Issue Tracker (пункт «Issues» в меню на главной странице репозитория), с описание найденных ошибок.

В поле *Title* вводится название ошибки, в поле *Description* — информация о найденной ошибке, т.е. здесь описываются шаги воспроизведения ошибки, результат, который ожидается в итоге, текущий результат. Пример: вы вводите в поле «Цена» отрицательное значение, но в итоге он получаете сообщение об ошибке, т. к. поле ввода содержит отрицательное значение. В поле «Assignee» необходимо выбрать имя разработчика из списка. Также можно задать приоритет в поле *Priority*, в зависимости от серьезности ошибки. Разработчик должен руководствоваться приоритетом при определении порядка исправления ошибок: сначала должны исправляться ошибки с наибольшим приоритетом. Также необходимо приложить скриншот ошибки, который поможет разработчику локализовать и исправить данную ошибку. Пример создания *issue* представлен на рис. 6.5.

## Issues

### Create issue

Title\*

Description

**H1 H2 H3 B I** ? Preview

What steps will reproduce the problem?  
1. In the "Price" field, you enter a negative value  
2.  
3.

What is the expected output? What do you see instead?  
To get a positive count of amount

Please use labels and text to provide additional information.  
To receive a message indicating that the task cannot be solved, because the input field contains a negative value

Assignee Philipp Stankevich (sta... x ▾)

Assign to me

Kind\*

Priority\*

Attachments

Рис. 6.5. Описание ошибки в Issue Tracker

6. Разработчик просматривает найденный дефект в Issue Tracker, можно воспользоваться пунктом меню My Issues (рис. 6.6). Затем разработкой проверяет описанный сценарий и исправляет ошибку в программе, выполняет Commit и Push изменений с помощью TortoiseGit (см. лабораторную работу №2). Затем разработчик заходит в Issue Tracker, нажимает кнопку «Resolve» (либо через меню «Workflow») (рис. 6.7–6.8), в поле комментариев вводит номер ревизии (номер внесённых изменений), см. пункт меню «Commits». Также можно выполнить редактирование *issue* с помощью кнопки «Edit» (рис. 6.8).

Issues + Create Issue

Filters: [All](#) [Open](#) [My issues](#) [Watching](#) Advanced search

Issues (1–2 of 2)

Title	T	P	Status	Votes	Assignee	Created	Updated ▾
#2: Start application problem			NEW		Philipp Stanke...	38 seconds ago	19 seconds ago
#1: Enter a title here			CLOSED		Philipp Stanke...	15 minutes ago	a minute ago

Рис. 6.6. My Issues

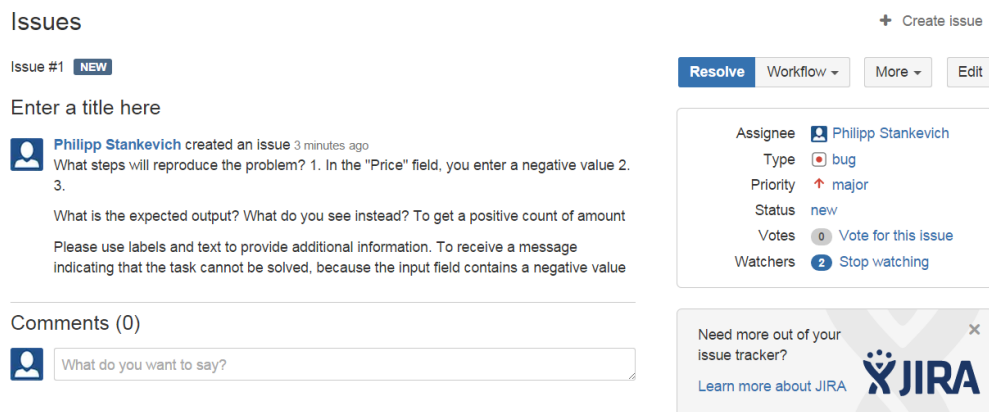


Рис. 6.7. Issue после создания

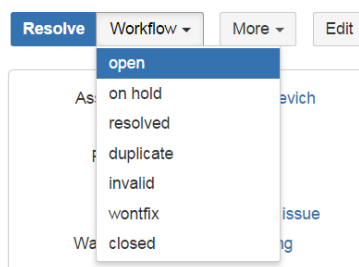


Рис. 6.8. Workflow

7. Специалист по тестированию обновляет свой локальный репозиторий (Pull) и запускает проект. Проверяет дефект, если он исправлен, то ставит статус Closed, в противном случае выполняют Open дефекта с соответствующим комментарием.

8. Специалисту по тестированию необходимо создать как минимум два дефекта в Issue Tracker, затем разработчик и специалист по тестированию меняются ролями.

### Контрольные вопросы

1. Что такое Issue Tracker? Где и для чего применяется?
2. Что является разновидностью Issue Tracker? (дать определение)
3. Какие сведения может включать в себя база данных об обнаруженных ошибках?
4. Привести пример жизненного цикла дефекта (с пояснениями).

### 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. Ron Patton. Software Testing, 2nd ed., Sams Publishing, 2006.
4. Савин Р. Тестирование dot com — М. : Дело, 2007 — 312 с.
5. Котляров В.П. Основы тестирования ПО — М. : Интернет-Ун-т Информ. Технологий, 2006. — 288 с.

## ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

Квалификационной работой по курсу «Основы промышленного программирования» должно стать индивидуальное домашнее задание (ИДЗ), демонстрирующее навыки, приобретенные студентом по итогам курса.

### Требования и рекомендации к ИДЗ

1. Задание должно быть выполнено в соответствии с принципами объектно-ориентированного программирования.
2. Должна быть создана и описана модель предметной области в т.ч. в виде диаграммы классов на языке UML.
3. Приложение должно быть разделено на модули (слои), такие как логика, графический интерфейс пользователя.
4. Приложение должно быть написано в соответствии с соглашением о кодировании.
5. Все исходные коды программы должны храниться в репозитории системы управления версиями, разработка должна вестись в репозитории в течении некоторого продолжительного промежутка времени, о чем должна свидетельствовать история коммитов (должно быть не менее трех осмысленных коммитов).
6. Переменные и классы (а также сам проект и его модули) должны носить осмысленные имена.
7. Рекомендуется в приложении использовать шаблоны проектирования.
8. Рекомендуется использовать журналирование событий.
9. Рекомендуется сложные для понимания места описывать в комментариях.

### Варианты индивидуальных заданий

1. Калькулятор.
2. Текстовый редактор, с функцией отмены последнего действия.
3. MP3-плеер.
4. Клиент для социальной сети (vk.com, facebook.com и т.п.).
5. Простой растровый редактор.
6. Программа для запоминания английских слов. Базовый словарь + возможность пополнения.
7. Простой растровый редактор.
8. Игра пятнашки.
9. Построитель графиков функций.
10. Счётчик строк кода (физических и логических, в т.ч. комментариев).

## СОДЕРЖАНИЕ

<i>О курсе</i> .....	3
Лабораторная работа № 1 Оформление текстового документа в Microsoft Office Word .....	4
Введение .....	4
1.1 Создание документа в Microsoft Office Word .....	4
1.2 Форматирование текста .....	4
1.3 Сноски и оглавления .....	9
1.4 Создание разделов .....	10
1.5 Колонтитулы .....	11
1.6 Нумерация страниц .....	12
1.7 Форматирование текста на уровне символов .....	13
1.8 Форматирование текста на уровне абзацев .....	14
1.9 Форматирование по образцу .....	16
1.10 Форматирование страницы. Подложка .....	16
1.11 Вставка символов и формул .....	17
Задание .....	19
Контрольные вопросы .....	20
Список литературы .....	21
Лабораторная работа № 2 Системы контроля версий .....	22
Введение .....	22
2.1 Создание проекта на BitBucket .....	22
2.2 Система контроля версий Git .....	24
2.3 Начало работы с BitBucket .....	24
Задание .....	30
Контрольные вопросы .....	30
Список литературы .....	30
Лабораторная работа № 3 Создание объектной модели предметной области .....	31
Введение .....	31
3.1 Сущности в UML .....	31
3.2 Отношения между сущностями .....	34
3.3 Работа в StarUML .....	38
3.4 Создание DLL .....	40
3.5 Соглашение о кодировании .....	41
Задание .....	42
Контрольные вопросы .....	44
Список литературы .....	44
Лабораторная работа № 4 Создание логики приложения .....	45
Введение .....	45
4.1. Разделение приложения на слои .....	45
4.2. Детальное проектирование .....	46
Рефакторинг .....	47
Задание .....	50
Контрольные вопросы .....	53
Список литературы .....	53



Лабораторная работа № 5 Знакомство с WinForms. Создание графического интерфейса .....	54
Введение.....	54
5.1 Создание приложения Windows Forms .....	54
5.2 Добавление элементов управления.....	55
5.3 Задание свойств .....	55
5.4           Список элементов.....	56
Задание .....	60
Контрольные вопросы.....	63
Список литературы .....	63
Лабораторная работа №6 Issue tracker на примере BitBucket .....	64
Введение.....	64
6.1 Работа в BitBucket .....	64
6.2 Состав информации о дефекте.....	65
6.3 Жизненный цикл дефекта.....	66
6.4 Примеры систем отслеживания ошибок.....	66
Задание .....	67
Контрольные вопросы.....	70
Список литературы .....	70
<b>ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ.....</b>	<b>71</b>
Требования и рекомендации к ИДЗ.....	71
Варианты индивидуальных заданий.....	71