Министерство образования и науки Российской Федерации
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

004
Ч-391

№ 4691

# ЧЕЛОВЕКО-МАШИННОЕ ВЗАИМОДЕЙСТВИЕ

Методические указания к выполнению лабораторных работ для студентов 3 курса образовательной программы:
02.03.03 – Математическое обеспечение и администрирование информационных систем,
профиль – Математическое и программное обеспечение информационных технологий
факультета прикладной математики и информатики

2-е издание

НОВОСИБИРСК
2017

УДК 004.5(076.5)
Ч-391

Составители

*А.С. Саутин*, канд. техн. наук,

*Т.А. Гультяева*, канд. техн. наук, доцент

Рецензент *В.Г. Кобылянский*, канд. техн. наук, доц.

Работа подготовлена на кафедре теоретической
и прикладной информатики

## СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Лабораторный практикум состоит из шести лабораторных работ, ориентированных на получение навыков разработки интерфейсов для программного обеспечения (ПО). Материал к каждой лабораторной работе разделен на следующие разделы:

- методические указания к лабораторным работам;
- задания;
- контрольные вопросы.

Варианты заданий к лабораторным работам приводятся ниже и являются общими для всех лабораторных работ. Защита лабораторных работ предполагает демонстрацию выполненного задания и ответы на контрольные вопросы. Для проведения лабораторных работ по данному курсу используется программное обеспечение Microsoft Visual Studio.

В результате прохождения данного курса студент должен приобрести следующие знания:

- способ описания требования к программному обеспечению на основе сценариев использования;
- принципы проектирования пользовательских интерфейсов;
- инструментарии, который представляет Microsoft Visual Studio для разработки интефейсов windows- и веб-приложений;
- принципы технологии AJAX в разработке интерфейсов и способы ее использования в Microsoft Visual Studio.

Также студент должен овладеть следующими навыками:

- описывать сценарии использования, представлять их с помощью use-case диаграмм и диаграмм деятельности;
- проектировать пользовательский интерфейс в соответствии с основными принципами проектирования интерфейсов;
- использовать средства Microsoft Visual Studio для разработки интерфейсов программного обеспечения.

## Варианты заданий к лабораторным работам

В каждом варианте должны быть реализованы соответствующие сценарии.

1. **Интернет -магазин:** покупка товара, поиск товара, добавление нового товара в базу данных магазина, просмотр и обработка заказов покупателей, регистрация нового покупателя.

2. **Книжный каталог:** добавление новой книги, поиск книги по нескольким полям, бронирование книги, списание старых книг, регистрация пользователей каталога.

3. **Адресная книга:** добавление нового абонента, добавление категорий абонентов, поиск абонентов по нескольким полям, добавления администратора каталога (пользователей, которые имеют право редактировать данные адресной книги), редактирование данных абонента.

4. **Расписание занятий:** добавление новой группы, добавление занятий (с указанием названия предмета, времени, аудитории, группы, недели, преподавателя, типа занятия), просмотр списка занятий на выбранную дату, добавление списка преподавателей, поиск занятий по нескольким полям (предмет, преподаватель, группа, время, тип занятия).

5. **База студентов:** добавление новой группы, добавление нового студента, поиск студента по различным полям, добавления информации об оценках по различным предметам, отчисление студента.

6. **Прайс-лист фирмы:** добавление новой категории товаров, добавление нового товара, поиск товара по различным полям, добавление администратора прайс-листа (пользователей, которые имеют право редактировать прайс-лист), перемещение товара из одной категории в другую.

7. **База склада фирмы:** добавление нового товара на склад, списание товара, выдача товара, поиск товара по различным полям, изменение месторасположения товара на складе.

8. **Аптечная база:** прием заказа от клиента на изготовление раствора, продажа лекарства, списание просроченных лекарств, добавление новые лекарств в базу данных, поиск заказов по различным полям.

# ЛАБОРАТОРНАЯ РАБОТА № 1

**Цель работы:** формирование навыков по анализу предметной области и написанию сценариев использования.

## Методические указания

### Сценарии использования

Сценарий – это один из способов описания структуры задачи [2]. Это повествовательный рассказ о совершаемых действиях, это история, эпизод, происходящий в данных временных рамках и в данном контексте. Различные формы сценариев широко применяются при разработке программного обеспечения. Сценарии задач и взаимодействий обычно богаты характеристиками и обладают высокой реалистичностью.

Сценарии при разработке пользовательского интерфейса описывают взаимодействие пользователя (или типа пользователей) и системы. Обыкновенные сценарии обладают некоторыми серьезными ограничениями при попытке использовать их для проектирования пользовательского интерфейса. В них делается упор на реалистичность и детали, при этом на серьезные проблемы и общую организацию обращается недостаточно внимания. Сценарии включают в себя правдоподобные описания комбинаций отдельных действий и задач, поэтому часто бывает тяжело выделить и понять основную суть взаимодействия.

### Модели use case

Концепция моделей use-case впервые была применена для разработки ПО Айваром Якобсоном в качестве составной части его объектно-ориентированного подхода к программной инженерии. Успех модели оказался столь значительным, что со временем произошла интеграция элементов use-case практически во все основные методы объектно-ориентированного анализа и проектирования. Несмотря на то, что модель была разработана специально для проектирования объектно-ориентированного ПО, ничего особенно «объектно-ориентированного» в элементах use-case нет, поэтому их можно применять практически ко всем подходам к проектированию.

Элемент use-case – это ситуация, вариант использования, т. е. некоторый случай применения системы. По сути use-case это:

- обеспечение функциональности;
- сугубо внешняя точка зрения (принцип «черного ящика»);
- повествовательное описание;
- описание взаимодействия пользователя (в какой-то роли) и системы;
- завершенное и понятное пользователю применение системы.

Каждый элемент use-case описывает в повествовательной форме завершенное, хорошо определенное взаимодействие, имеющее ясную цель с точки зрения пользователя. При объектно-ориентированном подходе элементы use-case могут описывать взаимодействие с другими системами и оборудованием, а не только с живыми пользователями. Тем не менее, когда целью является разработка пользовательского интерфейса, можно ограничиться рассмотрением только тех элементов use-case, которые относятся к взаимоотношениям человека и системы.

**Сущностные элементы use case**

Сущностный элемент use-case – это структурированное повествование, выраженное на языке данной прикладной области и пользователей системы и содержащее упрощенное, обобщенное, абстрактное, не зависящее от технологии и реализации описание одной завершенной, наполненной смыслом и хорошо определенное с точки зрения пользователей задачи или взаимодействия. Предполагается, что пользователь играет определенную роль по отношению к системе, а в описании воплощаются цели и замыслы лежащего в его основе взаимодействия.

Сущностные элементы use-case строятся на основе целей и задач пользователя, а не на основе каких-то конкретных механизмов или этапов, ведущих к достижению этих целей. Некоторым кажется значимым включение целей пользователей в модели use-case, но это не должно быть связано с упрощениями, присущими сущностному моделированию.

При использовании подхода, ориентированного на удобство использования, в сущностных элементах use-case, являющихся структурированным описанием, можно выделить три части: изложение общих устремлений пользователя, выраженное в элементе use-case, плюс состоящее из двух частей описание, включающее в себя модель пользовательских устремлений и модель обязательств системы. Сущностные элементы use-case именуются, причем при помощи этих имен стараются выразить пользовательские намерения в условиях данного варианта использования. В соответствии с соглашением, предложенным Якобсоном, элемент use-case изображается в виде эллипса с именем элемента.

**Описание вариантов использования**

Функциональные требования к системе моделируются и документируются с помощью вариантов использования (use-case). *Вариант использования* (use-case) – связный элемент функциональности, предоставляемый системой при взаимодействии с действующими лицами. *Действующее лицо* (actor) – роль, обобщение элементов внешнего окружения системы, ведущих себя по отношению к системе одинаковым образом.

В контексте процесса управления требованиями варианты использования трактуются следующим образом (согласно Коберну [1]).

- Вариант использования фиксирует соглашение между участниками проекта относительно поведения системы.
- Вариант использования описывает поведение системы при различных условиях, когда система отвечает на запрос одного из участников, называемого основным действующим лицом.
- Основное действующее лицо инициирует взаимодействие с системой, чтобы добиться некоторой цели. Система отвечает, соблюдая интересы всех участников.

Варианты использования – это вид документации, применяемый в том случае, когда требуется сконцентрировать усилия на обсуждении принципиальных требований к разрабатываемой системе, а не на подробном их описании. Стиль их написания зависит от масштаба, количества участников и критичности проекта. В общем случае, рекомендуется придерживаться следующих правил:

- названия вариантов использования должны быть деловыми (нетехническими) терминами, имеющими значение для заказчика;
- каждый вариант использования должен представлять собой завершенную транзакцию между пользователем и системой, представляющую для первого некоторую ценность;
- хорошо написанный вариант использования легко читается и состоит из предложений, написанных в единой грамматической форме. На обучение чтению варианта использования не должно уходить больше нескольких минут.

Формат описания варианта использования (по Коберну [1]).

1. Имя-цель в виде краткой активной глагольной фразы.
2. Контекст использования – более длинное описание цели.
3. Область действия.

4. Уровень точности.

5. Основное действующее лицо.

6. Другие участники и их интересы.

7. Предусловие (определяет, выполнение какого условия гарантирует система перед тем, как разрешить запуск варианта использования).

8. Минимальные гарантии (наименьшие обещания системы участникам, в частности, когда цель основного действующего лица не может быть достигнута).

9. Гарантии успеха (или постусловие – postcondition – устанавливает, что интересы участников удовлетворяются при успешном завершении варианта использования в конце основного сценария).

10. Триггер (событие, которое запускает вариант использования).

11. Основной сценарий или поток (простой для понимания типичный сценарий, в котором достигается цель основного действующего лица и удовлетворяются интересы всех участников). Каждый шаг основного сценария описывает взаимодействие двух действующих лиц («Клиент вводит адрес»); шаг подтверждения для защиты интереса участника («Система подтверждает PIN-код»); внутреннее изменение для удовлетворения интереса участника («Система выводит сумму из баланса»).

12. Расширения (запускаются при возникновении определенного условия, содержат последовательность шагов, описывающих, что происходит при этом условии, и заканчиваются достижением цели или отказом от нее).

13. Список изменений в технологии и данных.

14. Вспомогательная информация.

Шаблон RUP [5] для описания варианта использования похож на предлагаемый А. Коберном. Он отличается тем, что альтернативные потоки описываются отдельно от остальных расширений.

## Задания

1. Провести анализ предметной области в соответствии с выбранным заданием.

2. Составить пять сценариев использования программного обеспечения пользователем согласно формату описания Коберна.

3. Оформить отчет о проделанной работе.

4. Защитить лабораторную работу.

## Контрольные вопросы

1. Каким образом производится моделирование задач?
2. Что такое сценарий использования?
3. Что такое элемент use-case?
4. Что такое сущностные элементы use-case?
5. Чем отличаются сценарии использования от модели use-case?
6. Каким образом можно описать варианты использования?
7. Приведите пример описания варианта использования по Коберну?

# ЛАБОРАТОРНАЯ РАБОТА № 2

**Цель работы:** формирование навыков по построению use-case диаграмм и диаграмм деятельности.

## Методические указания

### Карты элементов use-case

Элементы use-case не существуют отдельно от внешнего мира. Полноценная программная система должна обеспечивать поддержку десятков, а то и сотен элементов use-case, причем, внутри каким-то образом связанного с ней приложения должна существовать связь между этими элементами . Отображение взаимосвязи приложений дает возможность описать общую структуру задачи, решаемой приложением и его интерфейсом. Карта элементов use-case для данной задачи разбивает все функциональные возможности системы на множество взаимосвязанных сущностных элементов use-case. Выделив все различающиеся и важные взаимодействия и показав отношения между ними, можно создать упрощенную общую модель задач, решаемых системой, и возможностей, которые она обязана предоставить.

Полноценная модель use-case представляет собой множество описаний, определяющих суть всех элементов use-case, и карту этих элементов, показывающую отношения между ними. Между сущностными элементами use-case могут существовать отношения разных типов, включая специализацию, расширение, композицию, а также сходство. Знание этих отношений позволяет аналитику или разработчику выделить общие элементы задачи и в итоге создать более простую модель задачи.

**Специализация**

Некоторые элементы use-case могут являться специализированными версиями других элементов. Например, при разработке приложения «банкомат» элементы use-case «получениеДенег», «размещениеСредств» и «запросСостояния» являются субклассами, или специализированными вариантами абстрактного класса взаимодействий, который может быть назван «использованиеБанкомата». Что касается отношения между элементами «получениеДенег» и «использованиеБанкомата», то его можно охарактеризовать как классификацию, или специализацию. Такой тип отношений означает, что один элемент use-case «является» («*is-a*») специализацией другого. В объектно-ориентированном анализе и проектировании такое отношение соответствует отношению класс–подкласс.

Специализация дает возможность упростить общую модель use-case путем отделения общих или универсальных форм взаимодействия от специфических форм, адаптированных для более узкого применения. Таким образом, нет необходимости переписывать заново самые общие паттерны применения. Достаточно написать их один раз, а затем лишь «повторно использовать» («reuse»), ссылаясь на них. Как видно из рис. 1, для отображения отношения специализации используется двойная стрелка. Рядом со стрелкой можно встретить подпись «is-a» или «specialize», в зависимости от контекста.

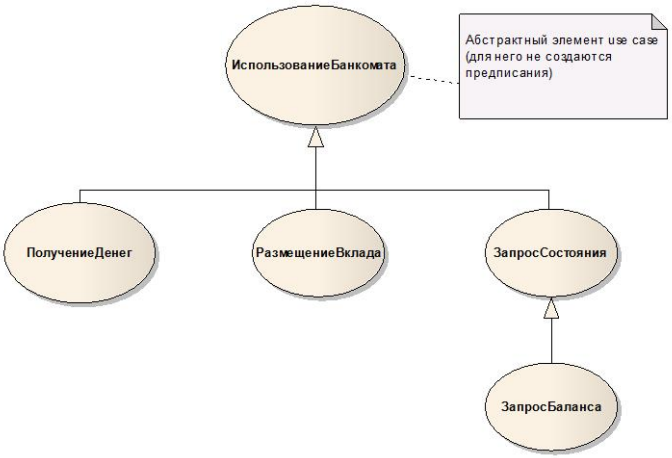


*Рис. 1.* Отношения специализации на карте элементов use case

**Расширение**

Одной из инноваций в объектно-ориентированной программной инженерии, навеянных идеями Якобсона, стало признание расширения одним из возможных отношений между элементами use-case. Говорят, что один элемент «расширяет» другой, когда он содержит вставляемые или альтернативные паттерны взаимодействия, которые войдут в расширяемый элемент. Например, при отработке элемента use-case для изменения внешнего вида какой-то части экрана пользователю необходимо заниматься поиском по всей системе файла, содержащего нужную картинку или значок. Нормальное, или ожидаемое, развитие событий (обеспечивается базисом, или базисным элементом use-case) тем не менее вовсе не подразумевает поиск каких-то дополнительных графических файлов.

Расширение – это удачная концепция, позволяющая значительно упростить сущностные модели use-case. На карте элементов use-case отношение расширения изображается в виде пунктирной линии со стрелкой и имеет подпись «extend», как показано на рис. 2. Если для расширения дается дополнительное описание, то в него может быть включено примечание, показывающее, какие элементы use-case расширяются.

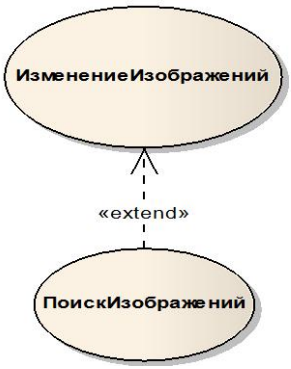


*Рис. 2.* Отношение расширения

**Композиция**

Элементы use-case можно декомпозировать на составные части, или подэлементы, являющиеся подчиненными или включенными пат-

тернами взаимодействия. Отношение композиции обозначается на карте элементов use-case пунктирной стрелкой, указывающей на подэлемент use-case и имеющей метку «include» как показано на рис. 3.

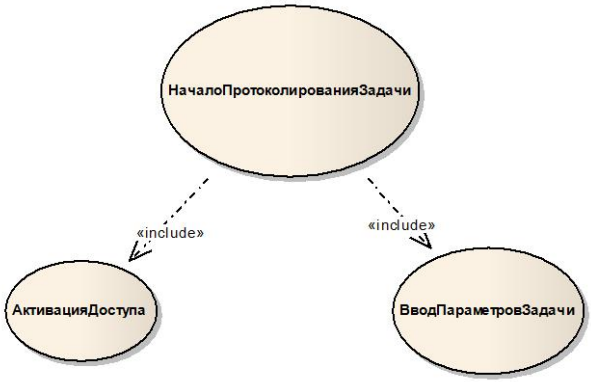


*Рис. 3.* Отношение композиции

Взаимодействие, описываемое суперэлементом use case, осуществляется при помощи взаимодействий, входящих в подэлемент или подэлементы, причем описание суперэлемента будет ссылаться на все используемые подэлементы. Например, элемент use-case под названием «началоПротоколированияЗадачи», созданный для программы, отслеживающей ход выполнения задач, может использовать элементы «авторизацияДоступа» и «вводПараметровЗадачи». Такой способ моделирования взаимодействий позволяет разделить независимые и почти никак не связанные между собой подзадачи «авторизацииДостула» и «вводаПараметровЗадачи».

**Диаграммы деятельности**

Для описания функциональных требований помимо диаграмм вариантов использования применяются диаграммы деятельности [4]:

- для описания поведения, включающего большое количество параллельных процессов;
- для анализа варианта использования (описывается последовательность действий и их взаимосвязь);
- для анализа потоков работ (workflow) в различных вариантах использования.

Когда варианты использования взаимодействуют друг с другом, диаграммы деятельности становятся средством представления и анализа их поведения.

Пример диаграммы деятельности представлен на рис. 4.

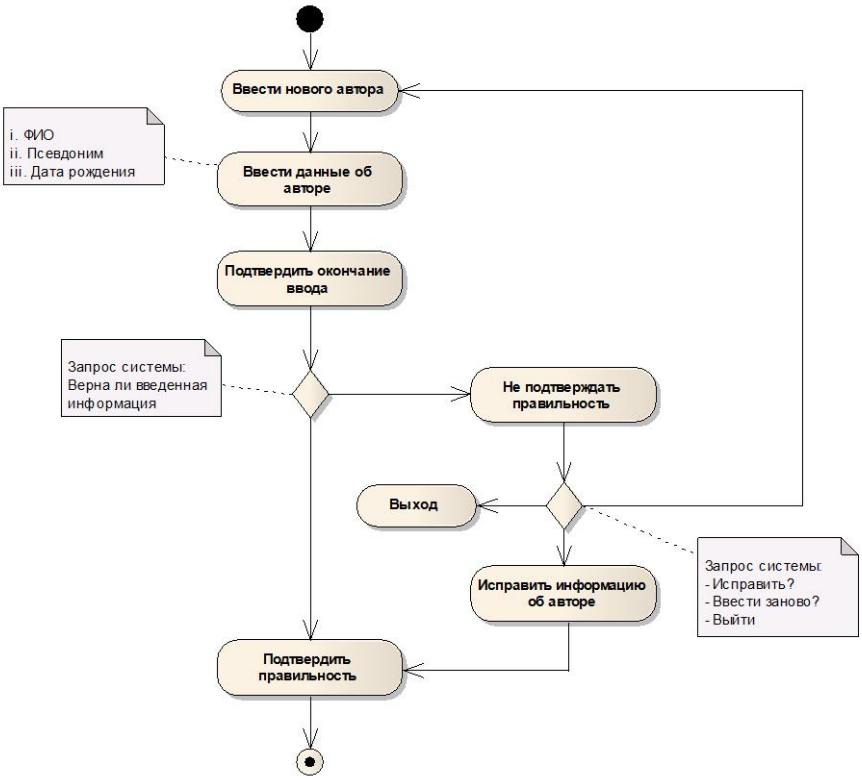


*Рис. 4*. Диаграмма деятельности

## Задания

1. Изучить основы построения use-case диаграмм и диаграмм деятельности.

2. Построить use-case диаграммы.

3. Построить диаграммы деятельности для каждого варианта использования из предыдущей лабораторной работы.

4. Оформить отчет о проделанной работе.

5. Защитить лабораторную работу.

## Контрольные вопросы

1. Что такое карта элементов use-case?

2. Что означает роль на use-case диаграмме?

3. В чем заключается суть отношения специализации? Приведите пример.

4. В чем заключается суть отношения расширения? Приведите пример.

5. В чем заключается суть отношения композиции? Приведите пример.

6. Чем отличается отношение специализации от расширения?

7. Что представляет собой диаграмма деятельности?

8. В чем заключаются отличия use-case диаграммы от диаграммы деятельности?

# ЛАБОРАТОРНАЯ РАБОТА № 3

**Цель работы:** формирование навыков создания прототипа интерфейса windows-приложения в соответствии с принципами проектирования пользовательского интерфейса.

## Методические указания

При создании интерфейса рекомендуется использовать существующие принципы проектирования пользовательского интерфейса [7]. Далее приводятся шесть принципов, вобравших в себя многое из того, что на данный момент известно о разработке эффективного пользовательского интерфейса. Каждый из них включает в себя несколько связанных между собой идей, более детализированных по сравнению с общими вопросами, к которым относят структуру, простоту, видимость, обратную связь, толерантность и повторное использование.

### Структурный принцип

*Организация пользовательского интерфейса должна быть целесообразной, осмысленной и удобной. Она должна базироваться на четких, целостных моделях, очевидных и распознаваемых пользователями.*

*При этом родственные понятия должны быть связаны, а независимые – разделены. Непохожие элементы должны дифференцироваться, а похожие – выглядеть похоже.*

Структурный принцип связан с общей архитектурой интерфейса и напрямую отражает представление о пользовательском интерфейсе как о диалоге между разработчиками и пользователями. Организация хороших интерфейсов продумывается очень тщательно, таким образом, чтобы отражать структуру решаемых системой задач и способ мышления пользователей относительно этих задач. Часто, особенно при использовании современных визуальных сред разработки , расположение визуальных компонентов внутри форм или диалогов и их распределение оказывается почти случайным и отражает в лучшем случае последовательность, в которой программистами затрагивались те или иные вопросы. По идее, свойства и функции , которые чаще всего используются совместно или рассматриваются пользователями как связанные друг с другом, должны располагаться вместе или, по крайней мере, быть четко и ясно взаимосвязанными. Что касается тех элементов, которые в контексте задачи или в сознании пользователя никак не связаны между собой, то они должны быть разнесены в интерфейсе. Подобное должно быть подобно. Похожая информация должна быть организована с помощью похожих решений, а объекты, обладающие похожим поведением, должны иметь общее представление.

**Принцип простоты**

*Следует максимально упрощать управление наиболее распространенными операциями. При этом общение с пользователем должно вестись на понятном для него языке. Должны предоставляться ссылки, логичным образом указывающие на более сложные процедуры.*

Процесс проектирования интерфейса – это всегда борьба за компромисс. Упрощение чего -то одного неизбежно приводит к усложнению чего-то другого. Если уменьшить количество меню, то увеличится число пунктов в каждом из них. Если сделать маленькими все диалоговые окна, включив в них как можно меньше элементов , любое взаимодействие пользователя с системой обернется для него необходимостью обращаться к большому количеству таких окошек.

Следование принципу простоты требует от вас знания задач, которые выполняются пользователем наиболее часто, и тех из них, которые, с точки зрения пользователя, проще. Именно такие задачи следует упрощать, чтобы пользователь мог быстро их решить.

**Принцип видимости**

*Все функции и данные, необходимые для выполнения данной задачи, должны быть видны, чтобы пользователь не отвлекался на дополнительную и избыточную информацию.*

Принцип видимости связан с проектированием таких пользовательских интерфейсов, в которых видны все элементы, необходимые для выполнения данной задачи. Цель – перейти от фил ософии WYSIWYG (What You See Is What You Get – что видишь на экране, то и получишь в результате) к философии WYSIWYN (What You See Is What You Need – на экране видишь то, что тебе нужно). Интерфейсы WYSIWYN оставляют видимыми только те элементы, которые действительно нужны пользователю для выполнения операции.

С одной стороны, в задачи проектирования входит создание такого интерфейса, на котором были бы явно видны все нужные и важные функции. С другой стороны, хороший интерфейс не должен загружать пользователя слишком большим количеством возможных вариантов или смущать его избыточной информацией. WYSIWYN-интерфейсы лучше уже тем, что они принимают во внимание ограниченность объема «оперативной памяти» человека и его способность узнавать вещи быстрее, чем вспоминать. Нагрузка на долговременную память уменьшается за счет того, что пользователь постоянно видит все необходимые опции и варианты. На кратковременную память нагрузка снижается за счет того, что пользователю не приходится запоминать и затем воспроизводить информацию, содержащуюся в какой-то другой части интерфейса.

**Принцип обратной связи**

*Сообщайте пользователям о действиях системы, ее реакциях, изменениях состояния или ситуации, об ошибках и исключениях, которые важны для них. Сообщения должны быть четкими, краткими, однозначными и написанными на языке, понятном пользователю.*

Хорошие пользовательские интерфейсы находятся в диалоге с пользователями, сообщая им о том, что происходит в системе. Принцип обратной связи указывает разработчикам на некоторые правила этого диалога.

Практичные системы информируют пользователя о множестве вещей. К примеру, они должны позволять ему узнавать о том, как воспринимаются вводимые им данные. Всякий раз, когда меняется внутреннее состояние системы, и это может оказать какое-либо влияние на

работу пользователя, его следует уведомлять об этом, особенно если меняется интерпретация системой его действий. Разумеется, пользователь должен знать о действиях, которые запрещены или игнорируются. При этом принцип обратной связи не может служить оправданием созданию бесконечных окошек сообщений. Информирование пользователя – не самоцель, а способ организации диалога в компактной и естественной форме.

Пользователям также требуются сообщения об ошибках и исключительных ситуациях. Во многих программах эти сообщения, к сожалению, неинформативны и способны ввести в заблуждение. Можно иногда встретить даже оскорбляющие сообщения. Или, скажем, такой текст: «Неправильно! Введите корректные данные!». Данное сообщение по сути дела, не дает никакой информации, так как, здесь не сказано, что именно неправильно и почему.

Грамотно составленные сообщения об ошибках – это еще один пример хорошей организации общения с пользователем. Рекомендации здесь могут быть следующие: краткость; язык, понятный пользователю; простота понимания. Прежде всего информативным должен быть заголовок сообщения. Он должен в сжатой форме описывать проблему, а уже само сообщение должно раскрывать подробности и предлагать способы решения или последовательность корректирующих действий.

**Принцип толерантности**

*Интерфейс должен быть гибким и толерантным. Ущерб, наносимый ошибками пользователя, необходимо снижать за счет возможности отмены и повтора действий, а также предотвращения появлений этих ошибок путем анализа различных форматов ввода и разумной интерпретации любых разумных действий.*

Интерфейс можно делать более или менее толерантным в зависимости от того, какие данные проверяются и когда. Проверка всех полей разом по окончании ввода данных – практика распространенная и иногда оправданная. При этом толерантности системе добавит автоматическая подсветка поля с неправильными данными, установка на нем курсора, плюс короткое, информативное сообщение в строке состояния. Больше всего пользователи страдают от программ, которые по окончании ввода во все поля проверяют всю форму. В случае неправильных данных хотя бы в одном из полей пользователь снова оказывается один на один с пустым бланком. Такое решение кажется неле-

пым, но встречается довольно часто, в том числе и в коммерческих программах.

В целом проверка неиспользуемых полей или полей, никак не обрабатываемых системой представляют интерес только для пользователей (в том виде, в каком они были введены) и ведет к снижению толерантности ПО. Например, проверка того, чтобы в поле примечаний присутствовали только буквенно-цифровые символы, избыточна. Кроме того, если пользователь захочет выделить что-нибудь в этом поле спецсимволом или с помощью псевдографики, то у него возникнут проблемы.

**Принцип повторного использования**

*Следует многократно использовать внутренние и внешние компоненты и принципы поведения системы, поддерживая устойчивость осмысленно, а не просто за счет избыточности. Это способствует уменьшению объема информации, которую пользователям приходится запоминать и о которой приходится думать каждый раз заново.*

Применяя повторное использование внешних и внутренних компонентов и решений, распространяющихся на всю систему, разработчик может создать не только более целостный интерфейс, но и более дешевый продукт. Стремление к одной лишь устойчивости повышает стоимость разработки, да и в ряде случаев оказывается сизифовым трудом. Нужно стремиться к устойчивости в контексте решаемых системой задач и области ее использования. Устойчивость ни в коем случае не может быть самоцелью, иначе она может привести к неудачным с точки зрения непротиворечивости системам.

Многие принятые стандарты и общепризнанные компоненты пользовательского интерфейса являют собой примеры неудавшихся попыток реализации устойчивости. Стандартные программные платформы порой навязывают разработчику плохо продуманные и неудачно спроектированные решения. Самые обычные диалоговые окна могут обеспечивать целостность и при этом быть весьма низкосортными, выбор стандартных горячих клавиш оказаться случайным и никак не соответствующим принципу устойчивости.

## Задания

1. Изучить среду программирования Visual Studio 2008.
2. Разработать прототип интерфейса windows-приложения в соответствии с основными принципами проектирования интерфейса (интерфейс должен быть достаточен для выполнения всех сценариев из лабораторной работы № 1).
3. Оформить отчет о проделанной работе, включающий в себя пример использования каждого из шести основных принципов проектирования интерфейса.
4. Защитить лабораторную работу.

## Контрольные вопросы

1. В чем заключается структурный принцип? Каким образом он использован в интерфейсе разработанной программы?
2. В чем заключается принцип простоты? Каким образом он использован в интерфейсе разработанной программы?
3. В чем заключается принцип видимости? Каким образом он использован в интерфейсе разработанной программы?
4. В чем заключается принцип обратной связи? Каким образом он использован в интерфейсе разработанной программы?
5. В чем заключается принцип толерантности? Каким образом он использован в интерфейсе разработанной программы?
6. Каким образом производится обработка событий для элементов интерфейса windows-приложения?
7. Каким образом следует проверять ошибки во введенных пользователем данных, и каким образом сообщать о них?

# ЛАБОРАТОРНАЯ РАБОТА № 4

**Цель работы:** формирование навыков создания веб-интерфейса в соответствии с принципами проектирования пользовательского интерфейса.

## Методические указания

ASP.NET файл является текстовым файлом и может содержать коды HTML, XML и языков сценариев [3, 6]. Коды последних выполняются на веб-сервере. Файл ASP.NET имеет специальное расширение ".aspx".

Порядок работы ASP.NET выглядит следующим образом:

- Когда веб-браузер запрашивает файл ASP.NET, веб-сервер IIS перенаправляет запрос модулю ASP.NET на сервер;
- Модуль ASP.NET читает файл построчно и выполняет, коды сценариев, содержащиеся в файле;
- Веб-браузеру возвращается обратно файл ASP.NET, но уже в виде обычного HTML документа.

Любая страница ASP.NET представлена классом, производным от класса System.Web.UI, который определяет свойства, методы и события, общие для всех страниц, предназначенных для обработки средой ASP.NET

Наиболее важные свойства этого объекта приведены в табл. 1:

Таблица 1

| Свойство | Описание |
|---|---|
| Application | Возвращает объект HttpApplicationState |
| Cache | Возвращает объект Cache, в котором хранятся данные приложения, в том числи и самой страницы |
| IsPostBack | Возвращает значение, определяющее, была ли страница загружена клиентом впервый раз, или загружена повторно в ответ на запрос клиента |
| Request | Возвращает объект HttpRequest, используемый для получения информации о входящем запросе HTTP |
| Response | Возвращает объект HttpResponse, используемые для формирования ответа сервера клиенту |
| Server | Возвращает объект HttpServerUtility |
| Session | Возвращает объект System.Web.SessionState.HttpSessionState, с помощью которого выдается информация о текущем сеансе HTTP |

Такое построение проекта позволяет хранить отдельно код представления для генерации HTML кода (в файле *.aspx) от программной логики (в файле *.aspx.cs), что во многих случаях существенно упрощает разработку сложных веб-приложений.

**Порядок создания приложения на ASP.NET**

1. Создание нового проекта в среде Microsoft Visual Studio с использованием шаблона ASP.NET Web Application.

2. Каждое из веб-приложений для IIS должно размещаться в своем виртуальном каталоге, которому соответствует физический каталог на диске. Для создания виртуального каталога необходимо:

– открыть оснастку Microsoft Internet Information Services (Пуск > Панель управления > Администрирование > Internet Information Services);
– раскрыть ветку «Веб-узлы» и, перейдя в «Веб-узел по умолчанию», создать новый виртуальный каталог;
– в свойствах созданного виртуального каталога выбрать закладку «Документы» и добавить в список документов, используемых по умолчанию, документ Default.aspx.

3. После компиляции проекта (опция меню «Build» или <Shift+F6>) и его выполнения (<Ctrl+F5>) в браузере можно будет увидеть результат выполнения сценария.

После завершения создания проекта, он будет содержать файлы Default.aspx, Default.aspx.cs и Default.designer.cs.

**Серверные элементы управления ASP.NET**

Важной особенностью ASP.NET является использование *серверных элементов* управления на веб-странице (элементы WebForm), которые являются фактически тэгами, понятными веб-серверу. Эти элементы определены в пространстве имен System.Web.UI.WebControls.

Принято выделять три типа серверных элементов управления:

- серверные элементы управления HTML – обычные HTML тэги;
- элементы управления веб-сервера – новые тэги ASP.NET;
- серверные элементы управления для проверки данных (валидации) – применяются для валидации входных данных от клиентского приложения (обычно веб-браузера).

Преимущества от использования таких элементов при разработке веб-приложений:

- сокращается количество кода, написанного вручную (что особенно заметно в для сложных элементов документа). Элемент просто «перетаскивается» из панели инструментов, после чего выполняется настройка его параметров в специальном окне. При этом все изменения автоматически заносятся непосредственно в *.aspx файл;
- с программной точки зрения каждому из этих элементов управления соответствует определенный класс в библиотеке базовых классов .NET, что позволяет писать для них такой же код как и для любых других классов;

- для любого элемента управления WebForm определен набор событий, обрабатываемых на веб-сервере;
- для любого элемента управления WebForm предоставляется возможность проверки ввода данных пользователем.

По умолчанию серверные элементы управления HTML в ASP.NET файлах рассматриваются как текст. Для их программирования требуется добавление атрибута runat="server" в соответствующий HTML элемент. Кроме того, все серверные элементы управления HTML должны быть размещены внутри области действия тэга <form>, также имеющего атрибут runat="server".

Подобно серверным элементам управления HTML элементы управления веб -сервера также создаются на веб -сервере и предполага-ют добавление атрибута runat="server". Однако они могут и не соот-ветствовать конкретным элементам HTML, но представлять более сложные элементы.

Общий синтаксис для описания таких элементов:

```
<asp:тип_элемента id="идентификатор" runat="server"/>
```

Серверные элементы валидации применяются для проверки вводимых пользователем данных и имеют аналогичный синтаксис.

**Серверные элементы управления для проверки данных (валидации)**

Элементы управления данного типа применяются для проверки вводимых данных и имеют следующий синтаксис:

```
<asp:тип_элемента id="идентификатор" runat="server" />
```

Наиболее важные элементы приводятся в следующей табл. 2.

Таблица 2

| Элемент управления для проверки данных | Описание |
|---|---|
| CompareValidator | Сравнивает значение, введенное в один элемент управления со значением, введенным в другой элемент, либо с фиксированным значением |
| CustomValidator | Позволяет задавать пользовательский метод проверки вводимых значений |
| RangeValidator | Проверяет, что значение, введенное пользователем, находится между двумя величинами |

Окончание табл. 2

| Элемент управления для проверки данных | Описание |
|---|---|
| RegularExpressionValidator | Проверяет введенное значение на соответствие указанному шаблону |
| RequiredFieldValidator | Проверяет обязательное наличие введенного значения |
| ValidationSummary | Отображает отчет обо всех ошибках проверки значений, произошедших на веб-странице |

**Элементы управления веб-сервера**

Перечень доступных элементов такого типа приведен в табл. 3.

Таблица 3

| Элемент управления веб-сервера | Описание |
|---|---|
| AdRotator | Отображение случайно выбранных рекламных объявлений |
| Button | Отображение кнопки |
| Calendar | Отображение календаря |
| CalendarDay | Элемент выбора дня календаря |
| CheckBox | Отображение флажка |
| CheckBoxList | Группа флажков |
| DataGrid | Отображение полей источника данных |
| DataList | Отображение элементов из источника данных с помощью шаблонов |
| DropDownList | Выпадающий список |
| HyperLink | Гиперссылка |
| Image | Изображение |
| ImageButton | Кнопка в виде изображения |
| Label | Отображение статического содержимого, доступного для программирования, и с возможностью задания стиля |
| LinkButton | Кнопка с гиперссылкой |
| ListBox | Выпадающий список с единичным или множественным выделением |
| ListItem | Элемент списка |
| Literal | Отображение статического содержимого, доступного для программирования, но без возможности задать стиль |
| Panel | Контейнер для других элементов управления |

Окончание табл. 3

| Элемент управления веб-сервера | Описание |
|---|---|
| PlaceHolder | Место для добавления элементов управления программным способом |
| RadioButton | Радио-кнопка |
| RadioButtonList | Группа радио-кнопок |
| BulletedList | Маркерный список |
| Repeater | Повторяемый список элементов |
| Style | Задание стиля элемента управления |
| Table | Таблица |
| TableCell | Ячейка таблицы |
| TableRow | Строка таблицы |
| TextBox | Поле для ввода текста |
| Xml | Отображение XML файла или результата XSLT преобразования |

## Задания

1. Изучить основы ASP.Net.
2. Реализовать прототип веб-интерфейса приложения средствами Visual Studio 2008 на ASP.Net (C#) на основе принципов проектирования интерфейса, описанных в предыдущей лабораторной работе. Интерфейс должен быть достаточен для выполнения всех сценариев из лабораторной работы № 1. Для хранения данных можно использовать СУБД MS SQL Server.
3. Оформить отчет о проделанной работе, включающий в себя пример использования каждого из шести основных принципов проектирования интерфейса.
4. Оформить отчет о проделанной работе.
5. Защитить лабораторную работу.

## Контрольные вопросы

1. В чем заключаются основные отличия веб-интерфейса от интерфейса windows-приложения?
2. Какими преимуществами обладает веб-интерфейса в сравнении с интерфейсом windows-приложения?

3. Какими недостатками обладает веб-интерфейса в сравнении с интерфейсом windows-приложения?

4. В каких случаях целесообразно применять веб-интерфейс?

5. Какие элементы интерфейса могут использоваться при построении веб-интерфейса?

6. Отличаются ли эти элементы веб-интерфейса от соответствующих элементов windows-приложения?

7. Каким образом производится обработка событий для элементов веб-интерфейса?

8. Какую роль играет HTML в построении веб-интерфейса?

9. Каким образом производится проверка вводимых пользователем данных в веб-приложении? В чем заключаются отличия данного способа проверки от проверки данных в windows-приложении?

# ЛАБОРАТОРНАЯ РАБОТА № 5

**Цель работы:** формирование навыков реализации бизнес-логики windows-приложения и веб-интерфейса.

## Методические указания

В данной лабораторной работе необходимо доработать предыдущие два типа интерфейсов с целью реализации полноценной работы пользователя с системой. При этом должны быть реализованы все возможные проверки данных вводимых пользователем. Функциональность приложений должна быть достаточна для выполнения всех вариантов использования из лабораторной работы № 1. Для хранения данных можно использовать СУБД MS SQL Server.

Для того чтобы создавать интересные web-страницы, необходимо наполнить их динамичным, обновляемым содержанием [3]. Особенно необходимо это в бизнес-приложениях – банковских, интернет-магазинах и аукционах. Важная часть работы, которую выполняет разработчик ASP.NET – это связывание своих страниц с источниками данных, отображение данных на странице, создание удобных средств взаимодействия с ними.

Для хранения данных чаще всего используются СУБД (системы управления базами данных). В ASP.NET 2.0 работа с данными происходит через ADO.NET 2.0 – часть .NET, разработанную специально для доступа к базам данных или XML-файлам.

Для работы с базами данных используется язык структурированных запросов – SQL (Structured Query Language). Команды этого языка называются запросами. Запросы служат для получения данных, для создания и изменения структуры таблиц, добавления, удаления и обновления записей и многого другого. Последовательность команд может храниться прямо на сервере СУБД в виде хранимой процедуры. Нужно стараться всегда пользоваться хранимыми процедурами, а не писать команды самим. Главное их преимущество – скорость работы и инкапсуляция бизнес-логики. Хранятся они на сервере в уже откомпилированном виде, в то время как простой переданный набор команд SQL проходит через стадию компиляции.

Для обращения к базам данных из внешних программ существуют специальные механизмы. В Windows это ODBC – открытый интерфейс взаимодействия с базами данных. Он позволяет приложениям, работающим под Windows или другими ОС, общаться с различными серверами реляционных баз данных.

Для конфигурирования источников данных зайдите в Control Panel, Administrative Tools, Data Sources (ODBC).

ODBC при наличии нужного драйвера позволяет связываться с различными базами данных – Access, FoxPro, Oracle, Microsoft SQL, MySQL, SAP, DB2. Если в файле Excel создать именованную таблицу, то ODBC способен ее распознать и работать как с таблицей базы данных.

ADO.NET – это набор классов для работы с внешними данными. В новой версии . NET 2.0 он был расширен новыми свойствами и тоже получил номер 2.0.

Соединение в ADO.NET может происходить с помощью различных провайдеров. В настоящее время рекомендуется работать с помощью провайдера MS SQL или Oracle. Эти провайдеры сами написаны на управляемом коде .NET. Еще один провайдер, OLEDB, позволяет получить доступ к другим источникам данных – Access, Excel, MySql, SAP. Провайдер OLEDB написан на неуправляемом коде, но может работать вместе с .NET.

Классы ADO .NET объединены в несколько пространств имен.

System.Data – это ядро ADO.NET. Оно содержит классы, необходимые для связи посредством любых провайдеров данных. Эти классы представляют таблицы, строки, столбцы, DataSet (множество взаимосвязанных таблиц). Там определены интерфейсы (в смысле языка C#) соединений с базами данных, команд, адаптеров данных.

System.Data.Common – базовые классы для всех провайдеров данных – DbConnection, DbCommand, DbDataAdapter.

В System.Data.OleDb находятся классы, позволяющие работать с источниками данных OleDb, в том числе с MS SQL версии 6.0 и ниже. Там находятся такие классы, как OleDbConnection, OleDbDataAdapter и OleDbCommand.

System.Data.Odbc содержит классы, которые работают с источниками данных ODBC посредством провайдера .NET ODBC. Классы имеют аналогичные имена с префиксом Odbc.

System.Data.SqlClient. Здесь определен провайдер данных для СУБД SQL Server версии 7.0 и выше . Содержатся классы SqlConnection, SqlTransaction, SqlCommand и др.

В System.Data.SqlTypes находятся классы, представляющие типы данных СУБД SQL Server.

Классы ADO .NET делятся на три типа. Классы типа Disconnected определяют базовую структуру данных, например, DataTable. Они независимы от каких-либо провайдеров данных и могут создаваться и заполняться данными непосредственно в программе. Классы Shared – базовые и общие для всех провайдеров. Классы Data Provider – специфические для разных провайдеров.

**Программирование ADO.NET**

Все провайдеры данных содержат классы соединений, адаптеров, команд. Схема типичной программы в ADO.NET следующая.

1. Вначале создается соединение с базой данных – класс Connection, который обеспечивается необходимой информацией – строкой соединения.

2. Создается объект Command и задается команда, которую необходимо выполнить в данной СУБД. Эта команда может быть запросом SQL или исполняемой процедурой. Нужно задать параметры этой команды, если они имеются.

3. Если команда не возвращает данных, то она просто выполняется с помощью одного из методов Execute. Например, это может быть удаление или обновление данных таблицы.

4. Если команда возвращает выборку данных, то их необходимо куда-то поместить. Решите, что вам нужно: получить данные для последующего использования без связи с базой данных или же просто быстро выполнить команду. В первом случае нужно создать класс DataAdapter и с его помощью сохранить данные в DataSet или в

DataTable. Во втором случае создается класс DataReader, который требует сохранять соединение на все время работы, хранит выборку только для чтения и позволяет двигаться только вперед. Зато чтение с помощью DataReader выполняется в несколько раз быстрее, чем в DataAdapter.

5. Задать полученный DataSet или DataReader как источник данных элемента управления или вывести их на страницу другим способом.

**Объект Connection**

Объект Connection для соединения с базой данных нуждается в строке соединения для указания пути к СУБД и входа в систему. Свойства класса Connection показаны в табл. 4. OleDbConnection, SqlConnection, OdbcConnection – наследники класса Connection, специфические для провайдеров OleDb, MS SQL ODBC соответственно.

Таблица 4

| Свойство | Описание |
|---|---|
| DataSource | Путь к базе данных в файловой системе при использовании Oledb, имя экземпляра базы сервера при использовании SqlConnection |
| Database | Возвращает имя базы данных, используемой в объекте Connection после открытия |
| State | Возвращает текущее состояние соединения. Возможные значения – Broken, Closed, Connecting, Executing, Fetching и Open |
| ConnectionString | Строка соединения с СУБД |

Все свойства, кроме ConnectionString, предназначения только для чтения.

**Использование объекта Command**

Объект Command исполняет запрос SQL, который может быть в форме встроенного текста, процедуры сервера или прямого доступа к таблице . Если это запрос на выборку данных SELECT, то данные обычно помещаются в DataSet или в DataReader. Методы и свойства определены в абстрактном классе DbCommand (через интерфейс IDbCommand), и их реализуют частные ненаследуемые классы OleDbCommand, SqlCommand, OdbcCommand.

Свойство *CommandType* может принимать значения из перечисления CommandType. По умолчанию это Text. То есть выполняется непосредственно текст команды SQL, который записан в свойстве Command. TableDirect означает, что в результате выполнения команды будет возвращено все содержание таблицы. StoredProcedure означает, что в Command находится имя процедуры сервера, которая и будет выполняться.

Свойство *CommandText* хранит текст запроса SQL или имя серверной процедуры.

*CommandTimeout* задает время ожидания ответа, по умолчанию равное 30 секундам. Если команда не выполнится в течение этого времени, будет выброшено исключение.

Процедуры сервера нуждаются в параметрах. Они хранятся в коллекции Parameters и имеют тип SqlParameter. Текстовые команды также могут получать параметры, перед которыми ставится префикс @. Например:

```
" SELECT * FROM CUSTOMERS WHERE CITY = @CITY AND CONTACTNAME = @CONTACT "
```

Часто используется метод *ExecuteNonQuery*. С помощью него можно выполнить любую операцию с базами данных, которая не связана с запросом и получением данных. Например, обновление, удаление записей, создание и изменение таблиц, создание процедур сервера. Она возвращает количество измененных записей в том случае, если выполняются команды Select, Update, Delete.

*ExecuteScalar* возвращает результат запроса в случае, если это одно-единственное значение. Например, нужно узнать количество заказов конкретного покупателя. Запрос выполняется с помощью команды "Select count * where customerid=1". Ее результат – выборка из одной строки и одного столбца. Ее можно выполнить и с помощью метода ExecuteReader, но ExecuteScalar будет выполняться быстрее. Если запрос возвратит большее количество строк или столбцов, то они будут проигнорированы.

*ExecuteRow* возвращает единственную запись.

*ExecuteReader* выполняется, если нужно получить табличные данные. Результат выполнения – курсор, в котором можно двигаться только от начала до конца.

В результате выполнения метода ExecuteReader объекта Command создается объект DataReader. Всегда закрывайте соединения после использования, иначе оно останется активным и будет занимать ресурсы.

Это можно сделать двумя способами. Первый – вызвать перегруженный метод ExecuteReader, который принимает параметр типа CommandBehavior со значением CommandBehavior.CloseConnection. В таком случае необходимо перелистать полученную выборку от начала до конца, и соединение закроется, когда будет достигнут конец. Если вы не хотите прочитать все данные, то можете самостоятельно закрыть соединение методом Close.

Развитые СУБД (теперь и MS Access) поддерживают транзакции. Транзакция – это последовательность команд, которая выполняется как одно целое. Например, при переводе денег сумма вычитается с одного счета и добавляется к другому. Если произойдет только одна из этих операций, то банк или его клиенты понесут потери. Поэтому важно, чтобы произошли обе операции либо не произошла ни одна. Если на одном из этапов транзакции допущена ошибка, то происходит откат (Rollback), т. е. отменяются все ранее сделанные операции и база возвращается к состоянию до начала транзакции. Если все успешно, то транзакция подтверждается операцией Commit.

Для поддержки транзакций введен класс SqlTransaction и ему подобные. У объекта Command есть свойство Transaction. Метод BeginTransaction объекта Connection заставляет базу данных перейти в режим транзакции.

Кроме того, необходимо всегда заключать программный код, работающий с базами данных, в блоки try/catch, поскольку работа часто идет с удаленными серверами и могут происходить самые разные ошибки как в сети, так и при работе самого сервера. При этом выбрасывается исключение SqlException или OleDbException:

**DataAdapter**

DbDataAdapter является родительским классом для SqlDataAdapter, OleDbDataAdapter, OdbcDataAdapter. Этот класс содержит четыре объекта типа Command. Классы DataAdapter обеспечивают двусторонний обмен информацией.

*SelectCommand* – используется для выборки данных из базы. При этом класс DataTable заполняется данными.

*UpdateCommand* – обновляет данные (редактирует записи).

*InsertCommand* – добавляет новые записи.

*DeleteCommand* – применяется для удаления записей.

Метод *Fill* класса DbDataAdapter заполняет объекты DataSet или DataTable данными, прочитанными в результате выполнения команды

SelectCommand. Эта команда должна быть запросом SQL типа Select. Если таблицы уже существуют, то в него добавляются новые таблицы. Вообще метод Fill перегружен восемь раз. Например, DbDataAdapter.Fill Method (DataSet, String) добавляет в DataSet таблицу с именем, указанным во втором параметре. Если такая таблица есть, то она обновляется. Доступ к таблице можно получить с помо-щью ее имени индексатором:

```
DataTable tblProducts = data.Tables["Products"];
```

Метод DbDataAdapter.Update записывает в базу данных все изменения, которые произошли в связанном с ним объекте DataSet.

**DataSet**

DataSet – это класс, содержащий одну или несколько таблиц DataTable и связи между ними. Класс DataSet – это представление в памяти информации, считанной через ADO из баз данных или XML. Он позволяет манипулировать данными после отключения от источника данных.

Коллекция таблиц хранится в свойстве Tables, а отношений – в свойстве Relations. Основываясь на таблицах DataSet, можно создавать представления – DataView.

## Задания

1. Изучить основы C# и работу с базами данных средствами .Net.
2. Реализовать в разработанных программах поддержку выполнения всех вариантов использования из лабораторной работы № 1.
3. Протестировать работу полученного приложения.
4. Оформить отчет о проделанной работе, отражающий пошаговое выполнений всех вариантов использования с помощью разработанных интерфейсов.
5. Защитить лабораторную работу.

## Контрольные вопросы

1. Каким образом можно работать с базой данных в .Net?
2. Чем отличается реализация бизнес-логики для веб-приложения и windows-приложения?
3. Что такое DataAdapter, какие методы он реализует?

4. Что такое DataSet и DataView?
5. Что такое ODBC?
6. Какие классы входят в ADO .NET?
7. Каким образом используется объект Command?
8. Каким образом можно отобразить данные в виде таблицы?

# ЛАБОРАТОРНАЯ РАБОТА № 6

**Цель работы:** формирование навыков по улучшению удобства использования веб-интерфейса приложения на основе технологии AJAX.

## Методические указания

В данной лабораторной работе необходимо добавить дополнительные проверки вводимых данных на стороне клиента с использованием технологии AJAX. Большинство обновляемых данных должно загружаться без перезагрузки веб-страницы целиком.

AJAX (Asynchronous JavaScript and XML) [5] – это концепция использования нескольких смежных технологий , ориентированная на разработку высокоинтерактивных приложений, быстро реагирующих на действия пользователя, выполняющих большую часть работы на стороне клиента и взаимодействующих с сервером посредством внеполосных обращений.

*Внеполосным обращением* называется запрос к серверу, который приводит к оперативному обновлению страницы вместо ее замены. Внеполосный вызов HTTP – это HTTP запрос, который выдается за пределами встроенного модуля, обеспечивающего отправку форм HTTP. Вызов инициируется событием, связанным со страницей HTML и обслуживается компонентом-посредником, обычно объектом *XmlHttpRequest*.

AJAX применяется для разработки веб-приложений, к которым предъявляются следующие требования:

- приложение должно передавать пользователям свежие данные, полученные с сервера;
- новые данные должны интегрироваться в существующую страницу без ее полного обновления.

Для работы с такими приложениями в браузере, необходимо, чтобы он соответствовал требованиям.

- Поддерживал посредников (для внеполосных вызовов HTTP). Обычно реализуется в форме объекта XmlHttpRequest.
- Поддерживал обновляемой модели DOM.

Объект *XmlHttpRequest* представляет собой компактную объектную модель для отправки сценарием обращений HTTP в обход браузера. Клиентский код сценария не может влиять на процесс размещения запроса и результат отправки запроса. XmlHttpRequest позволяет сценарию отправлять HTTP запросы и обрабатывать полученные ответы.

В качестве формата передачи данных обычно используются JSON или XML.

**Элементы управления сервера в Microsoft ASP.NET AJAX**

В ASP.NET 2.0 AJAX Extensions входят серверные элементы управления, используемые для частичных обновлений страниц, а также индикаторы выполнения, таймеры и компоненты управления скриптами.

Серверные элементы управления ASP.NET AJAX инкапсулируют как клиентское, так и серверное поведение. Далее представлен краткий обзор серверных элементов управления.

**ScriptManager**

Элемент управления ScriptManager управляет всеми клиентскими скриптами для ASP.NET AJAX. ScriptManager автоматически регистрирует скрипт для ASP.NET AJAX в момент его добавления на вебстраницу. Этот элемент должен быть первым в наборе элементов управления страницы. ScriptManager управляет частичным отображением страницы в браузере, если на странице находится один или несколько элементов управления UpdatePanel.

**UpdatePanel**

Элемент управления UpdatePanel сохраняет другие элементы управления и обеспечивает выполнение обновлений частей страниц. С помощью UpdatePanel элемента управления можно запрашивать обновления частей страницы без написания клиентского скрипта. Элементы управления внутри элемента управления UpdatePanel, которые обычно должны возвращаться для обновления своих данных, теперь довольно просто будут передаваться через обратный вызов типа AJAX, результатом которого будет скрытое обновление на сервере. Благодаря этому значительно упрощается взаимодействие приложения и элемента управления, поскольку отсутствуют события обратной передачи. Однако, если нужно включить основные сценарии и улучшить работу пользователей, то можно добавить настраиваемый клиентский скрипт. Отслеживание элемента управления UpdatePanel и связанных триггеров выполняет элемент управления ScriptManager.

**UpdateProgress**

Элемент управления UpdateProgress предоставляет сведения о состоянии обновлений частей страниц в элементах управления UpdatePanel. По умолчанию во время процесса обновления выполняется создание и отображение элемента «div». Настроить заданное по умолчанию отображение элемента управления «div» можно с помощью свойства ProgressTemplate.

**Timer**

Элемент управления Timer запускает обратную передачу через определенные интервалы времени. Этот элемент управления можно использовать для размещения всей страницы, а не обновлений части страницы. Элементы управления Timer используется внутри элемента управления UpdatePanel или вне его. Если нужно, чтобы элемент управления Timer запустил обновление, то к объявлению элемента управления UpdatePanel необходимо добавить триггер.

## Задания

1. Изучить основы технологии AJAX.
2. Добавить дополнительные проверки вводимых данных с использованием средств AJAX в приложение, созданное в предыдущей лабораторной работе.
3. Добавить возможность загрузки и редактирования данных без перезагрузки веб-страницы.
4. Протестировать работу полученного приложения.
5. Оформить отчет о проделанной работе.
6. Защитить лабораторную работу.

## Контрольные вопросы

1. Что такое технология AJAX?
2. Какая роль отводиться языку Javascript в технологии AJAX?
3. Какую роль играет XML в технологии AJAX?
4. В каких случаях целесообразно использовать технологию AJAX?
5. Какие преимущества дает технология AJAX по сравнению с обычным веб-интерфейсом?
6. Какими недостатками обладает технология AJAX?
7. Каким образом можно использовать AJAX в ASP.Net?
8. Какие элементы управления сервера существуют в ASP.Net AJAX?

## Рекомендуемые источники

1. *Коберн А.* Современные методы описания функциональных требований к системам. – М. Лори, 2002. – 263 с.

2. *Константайн Л., Локвуд Л.* Разработка программного обеспечения. – СПб.: Питер, 2004. – 592 с.

3. Основы ASP.NET 2.0 [Электронный ресурс]. – Режим доступа : http://www.intuit.ru

4. *Фаулер М.* UML. Основы. – СПб.: Символ-Плюс, 2004. – 192 с.

5. *Якобсон А., Буч Г., Рамбо Дж.* Унифицированный процесс разработки программного обеспечения. – СПб.: Питер, 2002. – 496 с. – Гл. 6, 7.

6. Web-технологии [Электронный ресурс]. – Режим доступа : http://www.intuit.ru

7. *Купер А.* Интерфейс. Основы проектирования взаимодействия / Алан Купер, Роберт Рейман, Дэвид Кронин, Кристофер Носсел. – СПб: Питер, 2016. - 719 с.

**ЧЕЛОВЕКО-МАШИННОЕ ВЗАИМОДЕЙСТВИЕ**

**Методические указания**

**2-е издание**

Редактор *А.Ю. Кроних*
Выпускающий редактор *И.П. Брованова*
Компьютерная верстка *Л.А. Веселовская*

---

Подписано в печать 08.06.2017. Формат 60 × 84 1/16. Бумага офсетная. Тираж 60 экз.
Уч.-изд. л. 2,09. Печ. л. 2,25. Заказ № 808/5. Цена договорная

---

Отпечатано в типографии
Новосибирского государственного технического университета
630073, г. Новосибирск, пр. К. Маркса, 20