Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования «Амурский государственный университет»

Кафедра Информационных и управляющих систем
(наименование кафедры)

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ

Программно-техническое обеспечение
(наименование дисциплины)

Основной образовательной программы по направлению подготовки:

231000.68 «Программная инженерия»
по магистерской программе «Управление разработкой программного обеспечения»
(код и наименование направления)

Благовещенск 2011

УМКД разработан _____
(степень, звание, фамилия, имя, отчество разработчиков)
_____
_____

Рассмотрен и рекомендован на заседании кафедры

Протокол заседания кафедры от «____» _____ 201___ г. № _____

Зав. кафедрой _____ /_____ /
(подпись) (И.О. Фамилия)

УТВЕРЖДЕН

Протокол заседания УМСС _____
(указывается название специальности (направления подготовки))

от «____» _____ 201___ г. № _____

Председатель УМСС _____ /_____ /
(подпись) (И.О.Фамилия)

1. Рабочая программа учебной дисциплины

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Амурский государственный университет»

УТВЕРЖДАЮ
Проректор по учебной работе
_____ В.В. Проказин
«___» _____ 20__ г.

РАБОЧАЯ ПРОГРАММА

ПРОГРАММНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Направление подготовки 231000.68 «Программная инженерия»
по магистерской программе «Управление разработкой программного обеспечения»
Квалификация (степень) выпускника – магистр
Специальное звание – нет

Курс – 2
Лекции – нет
Практические (семинарские) занятия – 18 (час.)
Лабораторные занятия – 18 (час.)
Самостоятельная работа – 72 (час.)
Общая трудоемкость дисциплины – 108 (час.), 3 (з.е.)
Курсовая работа (проект) – нет
Составитель – А.В. Бушманов, доцент, канд. техн. наук
Факультет математики и информатики
Кафедра информационных и управляющих систем

Семестр – 3
Экзамен – нет
Зачет – 3

2011 г.

Рабочая программа составлена на основании Федерального государственного образовательного стандарта высшего профессионального образования по направлению подготовки 231000 «Программная инженерия» (квалификация (степень) «магистр»)

Рабочая программа обсуждена на заседании кафедры информационных и управляющих систем
«___» _____ 20___ г., протокол № _____
Заведующий кафедрой _____ А.В. Бушманов

Рабочая программа одобрена на заседании учебно-методического совета направления подготовки 231000.68 «Программная инженерия»
«___» _____ 20___ г., протокол № _____
Председатель _____ В.В. Еремина

Рабочая программа переутверждена на заседании кафедры информационных и управляющих систем
«___» _____ 20___ г., протокол № _____
Заведующий кафедрой _____ А.В. Бушманов

СОГЛАСОВАНО
Начальник учебно-методического управления
_____
«___» _____ 20___ г.

СОГЛАСОВАНО
Председатель учебно-методического совета факультета
_____ С.Г. Самохвалова
«___» _____ 20___ г.

СОГЛАСОВАНО
Заведующий выпускающей кафедрой
_____ А.В. Бушманов
«___» _____ 20___ г.

СОГЛАСОВАНО
Директор научной библиотеки
_____ Л.А. Проказина
«___» _____ 20___ г.

1 ЦЕЛИ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Цель дисциплины – является обучение студентов методам разработки программ, а также структуры программного и технического обеспечения современных информационных систем.

Задачи дисциплины:

Выбора инструментальных программных средств;

Создания структуры приложения, папок ресурсов, файлов данных и файлов приложений;

Разработки оконных интерфейсов приложений;

Построения протоколов, программных интерфейсов и файлов реализации приложений.

2 МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ООП ВПО

Дисциплина относится к профессиональному циклу дисциплин (М2.В.ДВ.1) Федерального государственного образовательного стандарта высшего профессионального образования по направлению подготовки 231000.68 «Программная инженерия» (квалификация (степень) «магистр»).

Для успешного освоения данной дисциплины необходимы знания, умения и навыки, приобретенные в результате освоения дисциплин базовой части естественнонаучного цикла (Б.2) и дисциплин направления Федерального государственного образовательного стандарта высшего профессионального образования по направлению подготовки 231000 «Программная инженерия»: операционные системы; сети ЭВМ и телекоммуникации; программирование.

Знания, умения и навыки, приобретенные в результате освоения данной дисциплины необходимы для освоения дисциплин базовой и вариативной части профессионального цикла (М.2) Федерального государственного образовательного стандарта высшего профессионального образования по направлению подготовки 231000 «Программная инженерия» (квалификация (степень) «магистр»), а также прохождения научно-исследовательской практики и выполнения научно-исследовательской работы (М.3).

3 КОМПЕТЕНЦИИ ОБУЧАЮЩЕГОСЯ, ФОРМИРУЕМЫЕ В РЕЗУЛЬТАТЕ ОСВОЕНИЯ ДИСЦИПЛИНЫ

В результате освоения дисциплины обучающийся должен демонстрировать следующие результаты образования:

1) Знать:

- основные компоненты программного обеспечения;
- методы выбора инструментальных средств;
- методы создания структуры приложения;
- методы разработки интерфейсов приложений;
- методы разработки клиентских приложений, ориентированных на WEB;
- методы разработки приложений для платформ Мак и PC.

2) Уметь использовать:

- Современные инструментальные средства разработки:
- MS VisualStudio;
- Apple XCode
- Apple Dashcode
- DreamWeaver.
- Eclipse.

3) Владеть: навыками разработки приложений.

В процессе освоения данной дисциплины студент формирует и демонстрирует следующие общекультурные и профессиональные компетенции:

способность совершенствовать и развивать свой интеллектуальный и общекультурный уровень (ОК- 1);

умение отбирать и разрабатывать методы исследования объектов профессиональной деятельности на основе общих тенденций развития программной инженерии (ПК-1);

способность к проектной деятельности в профессиональной сфере на основе системного подхода, умение строить и использовать модели для описания и прогнозирования различных явлений, осуществлять их качественный и количественный анализ (ПК-6).

4 СТРУКТУРА И СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

Общая трудоемкость дисциплины составляет 3 зачетные единицы, 108 часа.

№ п/п	Раздел дисциплины	Семестр	Неделя семестра	Виды учебной работы, включая самостоятельную работу студентов и трудоемкость в часах				Формы текущего контроля успеваемости Форма промежуточной аттестации
				Лек	Пр	Лаб	Сам	
1	Назначение и основные возможности современных инструментальных средств. Этапы разработки программного обеспечения.	3	1-2	-	2	2	8	собеседование
2	Структура приложений. Программные среды для разработки локальных приложений.	3	3-4	-	2	2	8	Семинар
			5-6	-	2	2	8	Семинар
3	Построение консольных приложений. Построение приложений с использованием оконных интерфейсов.	3	7-8	-	2	2	8	Семинар
			9-10	-	2	2	8	Семинар
4	Model-View-Controller. Связь кода и элементов интерфейса. Примеры на C и Objective C. Элементы оконных интерфейсов.	3	11-12	-	2	2	8	Семинар
			13-14	-	2	2	8	Семинар
5	Построение многооконных приложений. Отладка приложений. Использование моделей данных	3	15-16	-	2	2	8	Семинар
			17-18	-	2	2	8	Семинар
6	Особенности построения приложений для WEB. Использование элементов GUI. CSS, HTML, Javascript, Ajax, Apache	3	1-18	-	18	18	72	Зачет

5 СОДЕРЖАНИЕ РАЗДЕЛОВ И ТЕМ ДИСЦИПЛИНЫ

5.1 Практические занятия

5.1.1 Практическое занятие 1. Назначение и основные возможности современных инструментальных средств. Этапы разработки программного обеспечения.

5.1.2 Практическое занятие 2. Структура приложений. Программные среды для разработки локальных приложений.

5.1.3 Практическое занятие 3. Построение консольных приложений. Построение приложений с использованием оконных интерфейсов.

5.1.4 Практическое занятие 4. Model-View-Controller. Связь кода и элементов интерфейса. Примеры на C и Objective C.

5.1.5 Практическое занятие 5. Элементы оконных интерфейсов.

5.1.6 Практическое занятие 6. Построение многооконных приложений. Отладка приложений.

5.1.7 Практическое занятие 7. Использование моделей данных.

5.1.8 Практическое занятие 8. Особенности построения приложений для WEB. Использование элементов GUI.

5.1.9 Практическое занятие 9. CSS, HTML, Javascript, Ajax, Apache.

5.2 Лабораторные работы

5.2.1 Лабораторная работа 1. Основы работы с VisualStudio. Отладка приложений в VisualStudio.

5.2.2 Лабораторная работа 2. Основы работы с XCode. Отладка приложений в XCode.

5.2.3 Лабораторная работа 3. Основы работы с DashCode. Отладка приложений в DashCode.

5.2.4 Лабораторная работа 4. Основы работы с Eclipse.

5.2.5 Лабораторная работа 5. Построение консольных приложений

5.2.6 Лабораторная работа 6. Особенности построения приложений для WEB

5.2.7 Лабораторная работа 7. Построение приложений с использованием оконных интерфейсов

5.2.8 Лабораторная работа 8. Использование элементов GUI

5.2.9 Лабораторная работа 9. Использование визуальных эффектов

6 САМОСТОЯТЕЛЬНАЯ РАБОТА

№ п/п	Раздел дисциплины	Форма (вид) самостоятельной работы	Трудоемкость в часах
1	Назначение и основные возможности современных инструментальных средств	Оформление отчета.	6
2	Структура приложений. Программные среды для разработки локальных приложений.	Оформление отчета.	12
3	Построение консольных приложений. Построение приложений с использованием оконных интерфейсов.	Оформление отчета.	12
4	Model-View-Controller. Связь кода и элементов интерфейса. Примеры на C и Objective C. Элементы оконных интерфейсов.	Оформление отчета.	12
5	CSS, HTML, Javascript, Ajax,	Оформление отчета.	26

	Apache		

7 МАТРИЦА КОМПЕТЕНЦИЙ

№ п/п	Раздел дисциплины	Общее кол-во компетенций			
		ОК1	ПК1	ПК6	
1	Назначение и основные возможности современных инструментальных средств. Этапы разработки программного обеспечения.	+			1
2	Структура приложений. Программные среды для разработки локальных приложений.	+			1
3	Построение консольных приложений. Построение приложений с использованием оконных интерфейсов.		+	+	2
4	Model-View-Controller. Связь кода и элементов интерфейса. Примеры на C и Objective C. Элементы оконных интерфейсов.		+	+	2
5	Построение многооконных приложений. Отладка приложений. Использование моделей данных		+	+	2
6	Особенности построения приложений для WEB. Использование элементов GUI. CSS, HTML, Javascript, Ajax, Apache	+		+	2

8 ОБРАЗОВАТЕЛЬНЫЕ ТЕХНОЛОГИИ

Образовательный процесс по дисциплине строится на основе комбинации следующих образовательных технологий.

Интегральную модель образовательного процесса по дисциплине формируют технологии методологического уровня: модульно-рейтинговое обучение, технология поэтапного формирования умственных действий, технология развивающего обучения, элементы технологии развития критического мышления.

Реализация данной модели предполагает использование следующих технологий стратегического уровня (задающих организационные формы взаимодействия субъектов образовательного процесса), осуществляемых с использованием определенных тактических процедур:

– лекционные (вводная лекция, информационная лекция, обзорная лекция, лекция-консультация, проблемная лекция);
– практические (углубление знаний, полученных на теоретических занятиях, решение задач);
– тренинговые (формирование определенных умений и навыков, формирование алгоритмического мышления);
– активизации познавательной деятельности (приемы технологии развития критического мышления через чтение и письмо, работа с литературой, подготовка презентаций по темам домашних работ);
– самоуправления (самостоятельная работа студентов, самостоятельное изучение материала).

Рекомендуется использование информационных технологий при организации коммуникации со студентами для представления информации, выдачи рекомендаций и консультирования по оперативным вопросам (электронная почта), использование при проведении лекционных и практических занятий мультимедиа-средств.

9 ОЦЕНОЧНЫЕ СРЕДСТВА ДЛЯ ТЕКУЩЕГО КОНТРОЛЯ УСПЕВАЕМОСТИ, ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ПО ИТОГАМ ОСВОЕНИЯ ДИСЦИПЛИНЫ И УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

9.1 Оценочные средства для текущего контроля успеваемости.

9.1.1 Контрольные вопросы допуска к выполнению реферативных работ.

9.1.2 Отчеты о выполнении индивидуальных вариантов заданий.

9.2 Оценочные средства для промежуточной аттестации

Вопросы к зачету:

9.2.1 Современные инструментальные среды – поддержка языков программирования и библиотек. Автоматизация процесса разработки ПО.

9.2.2 Жизненный цикл программного обеспечения. Разработка ПО. Отладка приложения. Подготовка релиза. Инсталляция ПО. Сопровождение ПО.

9.2.3 . Этапы работы приложения. Загрузка, создание визуального интерфейса, обработка событий. Файловая структура приложения. Ресурсы приложения. Пользовательский интерфейс. Многоязычная поддержка.

9.2. Возможности современных инструментальных систем по созданию приложений. Основные этапы создания приложения. Использование стандартных библиотек и системных ресурсов.

9.2.5 Пример приложения. Структура приложения. Подключение библиотек. Отладка приложения. Установка точек останова. Анализ информации от отладчика. Подготовка приложения к релизу.

9.2.6 Элементы GUI, используемые при построении оконных приложений. Соединение кода с событиями. Цикл обработки событий

9.2.7 Классическая схема построения оконных приложений. Представление, контроллер, модель. Связывание программного кода и событий. Использование имеющихся классов. Наследование.

9.2.8 . Представления, элементы для ввода-вывода информации, окна, контроллеры, элементы для вывода графики, видео и изображений. Другие элементы.

9.2.9 Планирование отладки. Использование точек останова. Анализ состояния программы. Просмотр значений переменных и объектов.

9.2.10 . Тестирование. Подготовка ресурсов. Создание многоязычных приложений.

9.2.11 Использование стандартного программного обеспечения для создания инсталлятора ПО.

9.2.12 Технические характеристики браузеров. Поддержка браузерами современных стандартов. Различия браузеров.

9.2.13 Стандартные элементы графического интерфейса пользователя. Моделирование интерфейсов локальных приложений.
9.2.14 Использование графики для создания «пользовательских элементов» GUI.
9.2. 15 Специальные библиотеки для создания GUI. Создание многовидовых приложений.
9.2.16 CSS переходы и анимация. Управление анимацией. Специальные виды эффектов.
9.2.17 Приложения, использующие данные. Разделение кода и данных. Данные, хранимые локально.
9.2.18 Обзор современных решений.
9.2.19 Построение модели данных. Связь источника данных с потребителем. Источники (базы данных, XML, JSON, key-value Store). События, изменяющие данные.
9.2. Перспективы развития Инструментальных средств.
9.3 Учебно-методическое обеспечение самостоятельной работы
9.3.1 Карточки с заданиями и методическими указаниями по выполнению практических работ
9.3.2 СТО СМК 4.2.3.05-2011. Стандарт ФГБОУОВПО «АмГУ». Оформление выпускных квалификационных и курсовых работ (проектов), 2011. – 95 с.

10 УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

а) основная литература:

10.1 Кузнецов И.Н. Научное исследование : методика проведения и оформления/ И. Н. Кузнецов . -3-е изд., перераб. и доп.. -М.: Дашков и К, 2008.-458 с
10.2 Рузавин Г.И. Методология научного познания : учеб. пособие : рек. УМЦ/ Г. И. Рузавин. -М.: ЮНИТИ-ДАНА, 2009.-288 с.
10.3 Кожухар В.М. Основы научных исследований: учеб. пособие/ В.М.Кожухар, -М.:Дашков и К, 2010. -216 с.

б) дополнительная литература:

10.4 Шкляр М.Ф. Основы научных исследований: учеб. пособие/М.Ф.Шкляр. -2-е изд., М.: Дашков и К, 2008, 2009, -244 с.
10.5 Безуглов Г.И. Основы научного исследования: учеб.пособие для аспирантов и студентов-дипломников/ И.Г.Безуглов, В.В.Лебединский, А.И.Безуглов, -М.: Академический Проект, 2008. -195 с.
10.6 Бережнова Е.В. Основы учебно-исследовательской деятельности студентов: учеб.: доп. Мин.Обр. РФ/ Е.В.Бережнова, В.В. Краевский. -3-е изд., стер., -М.: Академия, 2007. -128 с.
10.7 Введение в историю методологии науки : науч. -метод. пособие для препод. и слушателей молодеж. науч.-исслед. центра: лекционный курс, метод. рук. и программа/ сост. Г. А. Караваев. -М.: Спутник+, 2007.-182 с.

в) периодические издания:

10.8 Изобретения стран мира
10.9 Вестник Дальневосточного отделения Российской Академии наук
10.10 Научно-техническая информация. Сер 2, Информационные процессы и системы

г) программное обеспечение и Интернет-ресурсы:

Свободно распространяемое программное обеспечение

№ п/п	Наименование ресурса	Характеристика
1	рее http://www.iqlib.ru	Интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия.

		Удобный поиск по ключевым словам, отдельным темам и отраслям знаний.
2	http://www.intuit.ru	Интернет-университет информационных технологий, в котором вобраны электронные и видео-курсы по отраслям знаний
3	http://amursu.ru	Сайт АмГУ, Библиотека – электронная библиотека АмГУ
4	http://www.biblioclub.ru	Электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике, а так же содержит материалы по точным и естественным наукам

11 МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

11.1 Лекционная аудитория, оборудованная мультимедийными средствами

11.2 Учебные кабинеты, оборудованные рабочими местами пользователей ЭВМ

12 РЕЙТИНГОВАЯ ОЦЕНКА ЗНАНИЙ СТУДЕНТОВ ПО ДИСЦИПЛИНЕ

Семестровый модуль дисциплины						
№ п/п	Раздел дисциплины	Виды контроля	Сроки выполнения (недели)	Максимальное кол-во баллов	Посещение, активность на занятиях	Максимальное кол-во баллов за модуль
1	Назначение и основные возможности современных инструментальных средств. Этапы разработки программного обеспечения.	ПР № 1	1-2	5	1	6
2	Структура приложений. Программные среды для разработки локальных приложений.	ПР № 2 ПР № 3	3-4 5-6	5 5	1 1	12
3	Построение консольных приложений. Построение приложений с использованием оконных интерфейсов.	ПР № 4 ПР № 5	7-8 9-10	5 5	1 1	12
4	Model-View-Controller. Связь кода и элементов интерфейса. Примеры на C и Objective C. Элементы оконных интерфейсов.	ПР № 6 ПР № 7	11-12 13-14	5 5	1 1	12
5	Построение многооконных приложений. Отладка приложений. Использование моделей данных.	ПР № 8	15-16	5	1	9

6	Особенности построения приложений для WEB. Использование элементов GUI. CSS, HTML, Javascript, Ajax, Apache	ПР № 9	17-18	5	1	9
7	Промежуточная аттестация	зачет	18	6	0	40
Итого						100

2 Краткое изложение программного материала

Практическая работа 1.

Назначение и основные возможности современных инструментальных средств. Этапы разработки программного обеспечения.

Эта методология проектирования соединяет в себе объектную декомпозицию, приемы представления физической, логической, а также динамической и статической моделей системы.

Типовой проект включает в себя следующие этапы разработки программного обеспечения:

- анализ требований к проекту;
- проектирование;
- реализация;
- тестирование продукта;
- внедрение и поддержка.

Анализ требований к проекту

На этом этапе формулируются цели и задачи проекта, выделяются базовые сущности и взаимосвязи между ними. То есть, создается основа для дальнейшего проектирования системы.

В рамках данного этапа не только фиксируются требования заказчика, но и проводится их формирование – клиентам подбирается оптимальное решение их проблем, определяется необходимая степень автоматизации, выявляются наиболее актуальные для автоматизации бизнес-процессы.

При анализе требований определяются сроки и стоимость разработки ПО, формируется и подписывается ТЗ на разработку программного обеспечения.

Проектирование

На основе предыдущего этапа проводится проектирование системы. Эта методология проектирования соединяет в себе объектную декомпозицию, приемы представления физической, логической, а также динамической и статической моделей системы.

Во время проектирования разрабатываются проектные решения по выбору платформы, где будет функционировать система языка или языков реализации, назначаются требования к пользовательскому интерфейсу, определяется наиболее подходящая СУБД. Разрабатывается функциональная спецификация ПО: выбирается архитектура системы, оговариваются требования к аппаратному обеспечению, определяется набор орг. мероприятий, которые необходимы для внедрения ПО, а также перечень документов, регламентирующих его использование.

Реализация

Данный этап разработки программного обеспечения организован в соответствии с моделями эволюционного типа жизненного цикла ПО. При разработке применяются экспериментирование и анализ, строятся прототипы, как целой системы, так и ее частей. Прототипы дают возможность глубже вникнуть в проблему и принять все необходимые проектные решения еще на ранних этапах проектирования. Такие решения могут затрагивать разные части системы: внутреннюю организацию, пользовательский интерфейс, разграничение доступа и т.д. В результата этапа реализации появляется рабочая версия продукта.

Тестирование продукта

Тестирование тесно связано с такими этапами разработки программного обеспечения как проектирование и реализация. В систему встраиваются специальные механизмы, которые дают возможность производить тестирование системы на соответствие требований к ней, проверку оформления и наличие необходимого пакета документации.

Результатом тестирования является устранение всех недостатков системы и заключение о ее качестве.

Внедрение и поддержка

Внедрения системы обычно предусматривает следующие шаги:

- установка системы,
- обучение пользователей,
- эксплуатация.

К любой разработке прилагается полный пакет документации, который включает в себя описание системы, руководства пользователей и алгоритмы работы.

Поддержка функционирования ПО должна осуществляться группой технической поддержки разработчика.

Ключевые вопросы:

1. Анализ требований к проекту;
2. Проектирование;
3. Реализация;
4. Тестирование продукта.

Литература:

1 Кузнецов И.Н. Научное исследование : методика проведения и оформления/ И. Н. Кузнецов . -3-е изд., перераб. и доп.. -М.: Дашков и К, 2008.-458 с

2 Рузавин Г.И. Методология научного познания : учеб. пособие : рек. УМЦ/ Г. И. Рузавин. -М.: ЮНИТИ-ДАНА, 2009.-288 с.

3 Кожухар В.М. Основы научных исследований: учеб. пособие/ В.М.Кожухар, -М.:Дашков и К, 2010. -216 с.

Практическая работа 2.

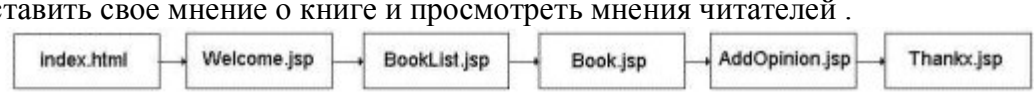
Структура приложений. Программные среды для разработки локальных приложений.

К сожалению, если сравнить два приложения - одно для Web, другое для обычной многопользовательской среды, первое оказывается всегда сложнее. Хотя бы взять наш пример. Что такое магазин. Заходит пользователь, выбирает товары и оплачивает покупки. Вся база данных будет состоять из 6-8 таблиц максимум. Если вы решили написать обычное приложение, то, в принципе, для этого не требуется очень глубокого анализа. Максимум времени, который вы потратите на разработку подобного приложения - две недели. Однако с распределенными приложениями все намного сложней - необходимо создавать пул соединений с базой данных, необходимо продумывать вопросы безопасности, авторизации пользователей и т.д. и т.п. Так что для начала необходимо продумать структуру приложения, причем сделать это достаточно аккуратно, что бы ни возвращаться к этому еще и еще.

Структура приложения.

В связи с тем, что мы делаем самые первые шаги в освоении новой технологии, не будем сильно усложнять задачу и начнем с того, что создадим несколько страничек нашего магазина, позволяющих пользователю:

- просмотреть список книг по темам;
- просмотреть предлагаемые книги по конкретно выбранной теме;
- оставить свое мнение о книге и просмотреть мнения читателей .



Исходя из этой структуры можно предположить следующее. Нам необходимо соединение с базой данных. Для этого мы воспользуемся классом DBConnectionManager , который

Вы можете найти по адресу http://webdevelopersjournal.com/columns/connection_pool.html. Там же есть небольшое описание, как пользоваться и пример использования. Кроме этого нам необходимы несколько классов, которые будут работать с книгами, присланными пожаланиями клиентов и т.д.

Сервлет Welcome.

Основное назначение этого сервлета - инициализация пула соединений. Конечно инициализацию можно было бы провести и по другому (например, в JSP странице), но так как целью статьи является обзор технологии, то мы воспользуемся сервлетом.

Любой сервлет имеет как минимум три метода – init(), service() и destroy(). В методе init() мы проведем создание класса DBConnectionManager –
DBConnectionManager conMgr = DBConnectionManager.getInstance();
В методе destroy() мы закроем все активные соединения с базой данных conMgr = DBConnectionManager.getInstance()
В методе service() проведем регистрацию созданного объекта под именем "conMgr" - getServletContext().setAttribute("conMgr", conMgr).
Таким образом у нас будет возможность получить ссылку на объект в любой момент пока активно наше приложение в JSP странице:
DBConnectionManager conMgr = (DBConnectionManager) application.getAttribute ("conMgr")

Класс Book.

В этом классе представлены методы для получения информации о книгах.

- bookGroupTitle - свойство для получения выбранной темы
- bookCount - количество книг по выбранной теме
- bookTable - таблица с выбранными книгами
- bookItem - единичный экземпляр книги

Установка всех этих свойств производиться в методах setBookTable и setBookItem. В JSP страницах значения свойств получаем посредством следующего вызова
<jsp:getProperty name="book" property="bookGroupTitle" />

Класс Opinion.

В этом классе представлены метод readersOpinions - мнения читателей. Установка свойства производится в методе setReadersOpinions.

JSP страницы.

Welcome.jsp.

Страница, где представлены темы для выбора и новости сервера. Переход на страницу Book.jsp осуществляется посредством передачи параметра bookGroupId. <a href="../html/BookList.jsp?bookGroupId=2"> (в случае с Java). Новости сервера формируются динамически из файлов news/Item1, 2... и включаются в JSP следующим образом <jsp:include page="news/Item1.html" flush="true" />

BookList.jsp.

Страница, где формируется таблица с книгами по выбранной теме. <jsp:useBean id="book" scope="session" class="Book" /> - создаем класс Book с идентификатором book. В связи с тем, что все наши классы будут иметь максимальную область видимости "session", то нет необходимости задумываться о синхронизации потоков.

Book.jsp.

Страница, где формируется сведения о конкретной книге. Кроме этого формируется список с мнениями читателей, если они есть.

AddOpinion.jsp.

Страница, где пользователь может высказать свое мнение о книге. Для этого мы будем использовать форму и два скрытых поля, что бы идентифицировать запрос
<input type="hidden" name="bookId" value="<%= bookId %>">
<input type="hidden" name="thankx" value="opinion">

Thankx.jsp

После отправки формы вызывается страница Thankx.jsp, которая будет использоваться для различных "спасибо" и ее содержание будет изменяться в зависимости от значения параметра Thankx. В нашем случае мы используем метод addOpinion класса Opinion.
Одно замечание по поводу перекодировки. Т.к. мы заставили пользователя использовать кодировку Cp1251, в каждой JSP странице есть строка <%@ page contentType="text/html; charset=windows-1251" %> то следующий процесс перекодировки верен и мы можем спокойно пользоваться этим обстоятельством в Opinion.java String opinion = new String(req.getParameter("opiniontext").getBytes("ISO-8859-1"),"Cp1251")
Чего не хватает в приложении?

- обработки ошибок
- для полного приложения необходимо предусмотреть ведение заказов покупателей и их оформление.

Надеюсь, что в продолжении статьи мы рассмотрим все эти вопросы и наше приложение будет полностью отвечать всем требованиям виртуального магазина.

Приложения

Таблицы базы данных.

Данное приложение было построено с использованием сервера Oracle, поэтому файл crttbl.sql использует диалект этого сервера. Я думаю, что не составит сложностей перевести его на другой сервер.

Менеджер соединений.

Его тоже необходимо настроить. Для этого существует файл [TOMCAT_HOME]\ webapps\bookshop\WEB-INF\classes\db.properties со следующим содеожанием

- drivers=oracle.jdbc.driver.OracleDriver - используемый вами драйвер
- logfile=D:\\Tomcat\\webapps\\bookshop\\log\\log.txt - место, где будет размещаться log файл
- guest.url=jdbc:oracle:thin:@192.37.3.58:1521:UPFR - URL вашего JDBC соединения
- guest.maxconn=50 - максимальное колличество соединений
- guest.user=scott - имя пользователя
- guest.password=tiger - пароль

В дальнейшем вы будете использовать имя guest для соединений с базой данных. Вообще данный файл может находиться где угодно лишь бы этот путь был доступен через переменную CLASSPATH. В нашем случае TomCat добавляет путь [TOMCAT_HOME]\ webapps\bookshop\WEB-INF\classes\ во время запуска.

Ключевые вопросы:
1 Структура приложения;
2 Сервлет Welcome;
3 Структура приложения;
4 Приложения.

Литература:
1 Кузнецов И.Н. Научное исследование : методика проведения и оформления/ И. Н. Кузнецов . -3-е изд., перераб. и доп.. -М.: Дашков и К, 2008.-458 с
2 Рузавин Г.И. Методология научного познания : учеб. пособие : рек. УМЦ/ Г. И. Рузавин. - М.: ЮНИТИ-ДАНА, 2009.-288 с.
3 Кожухар В.М. Основы научных исследований: учеб. пособие/ В.М.Кожухар, -М.:Дашков и К, 2010. -216 с.

Практическая работа 3.

Построение консольных приложений. Построение приложений с использованием оконных интерфейсов.

Окно — основа основ любого GUI-приложения. За его создание (а также поведение, внешний вид и т.д.) отвечает класс org.gnome.gtk.Window. В качестве примера приведем полный код класса создания пустого окна. Далее при рассмотрении элементов управления мы будем приводить только соответствующие фрагменты кода. Их положение в общей структуре программы обычно не будет вызывать вопросов, и мы не будем заострять на этом моменте внимание. Тем не менее, полный исходный код будет представлен в конце дискуссии.

Создание пустого окна.

```
package com.example;

import org.gnome.gtk.Gtk;
import org.gnome.gtk.Window;

public class UselessGUIApp {
   private UselessGUIApp(){
      final Window mainWindow;
      mainWindow = new Window();
      mainWindow.showAll();
   }

    public static void main(String[] args) {
      Gtk.init(args);

      new UselessGUIApp();

      Gtk.main();
   }
}
```

Хорошо известно, что приложение начинается с вызова статического метода main(). Назначение Gtk.init(args) и Gtk.main() обсуждается в предыдущей статье. Если запустить это приложение, то можно увидеть абсолютно пустое окно, без меню, кнопок и даже без заголовка. Кнопка закрытия окна — и та не работает нормально: не завершает приложение, а только скрывает окно. Ситуация вполне ожидаемая, т. к. никаких манипуляций с окном, кроме создания и отрисовки, мы не проводили. Ниже мы рассмотрим, как заставить эту кнопку работать. Конструктор без параметров создает окно верхнего уровня. Типы окон определены в классе WindowType и могут принимать значения TOPLEVEL или POPUP. В первом случае создаваемая область, называемая «окном», уже имеет «признаки оконного приложения», т. е. кнопки сворачивания/разворачивания, закрытия, границы, заголовок и т. д. Но самое главное — это окно «видно» менеджеру окон. Для создания основного интерфейса приложения нужно использовать именно такие окна. Тип POPUP (всплывающее окно) предназначен, очевидно, для различных подсказок, меню и т. п. По умолчанию, он не имеет никаких из вышеперечисленных атрибутов и представляет из себя лишь пустую прямоугольную область. Создать его можно, если явно указать тип в конструкторе Window: new Window(WindowType.POPUP);

Рассмотрим некоторые методы класса Window. Самый главный из них, пожалуй, — это showAll(). Без него окно и его содержимое не прорисуется на экране. Принцип работы с окном в общем заключается в следующем: сначала окно создается (new Window()), ему присваиваются различные свойства (например, setTitle("Window's Title")), добавляются элементы (метод add()), и затем вызывается метод showAll(), чтобы все это отрисовать. Отделение создания окна от его отрисовки обусловлено тем, что если оно появится на экране в момент создания, а затем ему, например, изменят ширину, то пользователь увидит дергающееся

окошко. При использовании же showAll() GTK+ уже имеет всю необходимую информацию и знает что, где и в каком виде рисовать. Это, конечно, не значит, что вы обязаны установить все свойства окна перед его отрисовкой, или что параметры отрисованного окна нельзя изменить.

Окно обладает различными свойствами: заголовком (Title), именем (Name), размерами (Width, Height, SizeRequest), модальностью (Modal) и т. д. Практически у каждого такого свойства есть соответствующие сеттеры. Например, добавим в наше приложение заголовок и установим размеры:

```
mainWindow.setTitle("GUI Application Title");
mainWindow.setSizeRequest(300, 100);
```

В качестве особенности можно отметить наличие свойства Stick. Метод setStick(boolean) устанавливает, будет ли окно видно на всех рабочих столах Gnome, KDE или аналогичного окружения. Здесь уместно затронуть вопрос кросс-платформенности: в Windows концепция множественных рабочих столов распространения не получила, что будет влиять на внешний вид и поведение вашего приложения. В данном случае это едва ли критично, но подобные вещи следует иметь в виду. Теперь поговорим о размещении в окне чего-нибудь полезного. Для этих целей существует абстрактный класс org.gnome.gtk.Container. Методов у него немного: add(Widget child), remove(Widget child), getChildren(), setBorderWidth(int width). Их говорящие имена едва ли требуют пояснений. Уже рассмотренный нами класс Window также является потомком Container, однако не прямым: Window наследует org.gnome.gtk.Bin, aBin — Container. Класс Bin — это контейнер, который может хранить только один элемент. На практике это означает, что использовать, например, Window для создания разметки не получится. Для этого существуют классы типа VBox и HBox. Как правило, в качестве этого единственного хранимого виджета для Window выступают представители именно этих двух классов. Например, обычная кнопка (org.gnome.gtk.Button) также является потомком Bin, и в ней может быть текст и/или иконка, но опять же — в единственном экземпляре.

Различные кнопки, поля ввода, надписи необходимо как-то группировать и размещать в удобочитаемом виде. Думая над разметкой, следует учитывать, что оперировать придется в основном классами VBox и HBox. HBox — это, грубо говоря, таблица с одной строкой и бесконечно большим числом колонок, VBox — таблица из одной колонки и также неограниченным числом строк.

Классы VBox и HBox наследуются от абстрактного org.gnome.gtk.Box, в котором сосредоточена вся функциональность этих виджетов. Кроме конструкторов, они ничего не переопределяют. Класс Box предназначен для организации элементов в прямоугольной области или в таблице. Используется он лишь в трех классах: VBox, HBox и ButtonBox. Этот класс примечателен своими методами добавления виджетов. Они более гибкие, чем add(Widget child) класса Container, потомком которого он является. Метод packStart(Widget child) добавляет элемент в начало контейнера (сверху вниз в VBox и слева направо в HBox), packEnd(Widget child) — в конец (снизу вверх в VBox и справа налево в HBox). Наверное, это требуется немного пояснить: виджет, установленный первым вызовом packStart будет первым, вторым вызовом — вторым и т. д., при использовании packEnd() список как бы переворачивается и происходит то же, что и при packStart() — виджет, установленный первым вызовом, будет последним, вторым вызовом — предпоследним и т. д. packStart() и packEnd() имеют свои расширенные версии, в которых можно задавать, например, будет ли элемент расширяться до размеров ячейки или величину границы. Метод packStart(Widget child), по сути, является полным эквивалентом add(Widget child).

Теоретически можно создать сколь угодно сложную разметку, используя лишь VBox и HBox, потому как они также наследуют классу Widget и, следовательно, могут быть вложенными друг в друга. Подобным образом и создаются интерфейсы: дочерним элементом в окне устанавливается экземпляр VBox, в ячейки этого контейнера помещаются экземпляры HBox, в его ячейки — экземпляры VBox и т. д. Степень вложенности зависит от того, насколько

сложный требуется интерфейс: кому-то можно будет обойтись вообще без контейнеров, т. е. разместить одну кнопку прямо в окне. Делать так не запрещается, ибо кнопка — тот же виджет, что и VBox.

Давайте добавим в наше приложение контейнер и разместим в нем виджет класса Label, т. е. просто некую надпись, чтобы он не пустовал.

```
VBox vbox = new VBox(false, 1);
Label label = new Label("Label #1");
vbox.add(label);
mainWindow.add(vbox);
```

Не забудьте, конечно, добавить сам контейнер в окно, иначе никакой надписи не появится. Конструктор VBox принимает два параметра. Первый отвечает за распределение пространства между дочерними элементами. Значение true означает, что всем элементам будет отдано одинаковое пространство. Это не очень удобно: задать отступы, скажем, в методе packStart уже не получится — скорее всего что-то «съедет» или вообще скроется, поэтому чаще используется значение false. Второй параметр — это расстояние между ячейками в пикселях. Конструктор HBox имеет те же самые параметры.

Список поддерживаемых GTK+ контейнеров, конечно, не исчерпывается классами HBox и VBox. Как некоторую альтернативу им следует упомянуть класс org.gnome.gtk.Table. Как нетрудно догадаться по названию, этот контейнер распределяет элементы в виде таблицы. При создании необходимо указать количество строк и количество колонок. Добавляются виджеты методом attach(Widget child, int leftAttach, int rightAttach, int topAttach, int bottomAttach). Например, вызов attach(new Label("text"), 1, 2, 1, 2) добавит строку "text" в ячейку с координатами (2, 2). Однако использовать этот класс не рекомендуется, и даже в javadoc о нем написано (цитирую): "a pain in the ass to use". Гораздо удобнее пользоваться набором VBox и HBox.

В некоторых случаях может быть полезным такой контейнер, как org.gnome.gtk.Fixed. Его особенность — это возможность расположить элемент в любой точке окна. Элемент также можно перемещать. Для этого в нем описаны методы put(Widget widget, int x, int y) и move(Widget widget, int x, int y). Пояснять их, наверное, не требуется, напомню лишь, что координаты отсчитываются слева направо и сверху вниз.

Если ваши виджеты не помещаются на отведенное для них пространство, то они просто скрываются. Для организации скроллинга в библиотеке присутствует контейнер org.gnome.gtk.ScrolledWindow. Он, подобно org.gnome.gtk.Window, также наследуется от org.gnome.gtk.Bin.

Разделить окно на две области позволяет org.gnome.gtk.Paned, а точнее — его производные org.gnome.gtk.HPaned и org.gnome.gtk.VPaned. За добавление элементов в левую/верхнюю (HPaned/VPaned) и правую/нижнюю области отвечают методы add1(Widget child) и add2(Widget child).

Существуют также контейнеры для специфического содержимого. Это org.gnome.gtk.IconView, org.gnome.gtk.TextView, org.gnome.gtk.Toolbar и т. д. Например, Toolbar — хорошо известная всем панель инструментов, обычно располагающаяся под главным меню — позволяет добавлять в себя только виджеты класса org.gnome.gtk.ToolItem.

Создать окно с вкладками можно при помощи контейнера org.gnome.gtk.Notebook. У него, как и у всех, существует метод add(Widget child), но использовать лучше appendPage(Widget child, Widget tabLabel) или prependPage(Widget child, Widget tabLabel). Второй параметр позволяет озаглавить вкладку [Обратите внимание, что он имеет тип Widget, а не String. Используя HBox, можно разместить «на корешке» вкладки, например, пиктограмму, название и кнопку закрытия, как в популярных браузерах. — Прим. науч. ред.]. В качестве примера использования Notebook можно привести следующие строки. Их необходимо вставить в наше приложение взамен тех, где мы добавляли главный контейнер в окно.

```
Notebook nb = new Notebook();
```

```
        nb.appendPage(new Label("Label Content"), new Label("Label Title"));
        mainWindow.add(nb);
```

Этот код создаст одну вкладку с надписью "Label Title" и содержимым "Label Content". Добавить вторую вкладку также не сложно: можно лишь второй раз вызвать nb.appendPage(new Label("Label Content"), new Label("Label Title"));

Следует также отметить, что один и то же объект не может быть вложен в несколько контейнеров — при этом возникает исключение org.gnome.glib.FatalError с сообщением вроде "Attempting to add a widget with type GtkLabel to a container of type GtkVBox, but the widget is already inside a container of type GtkVBox" ("Попытка добавить виджет типа GtkLabel в контейнер типа GtkVBox, но виджет уже находится в контейнере типа GtkVBox"). Например, если одну и ту же надпись нужно разместить в нескольких местах, то придется создавать несколько объектов Label.

Полный код демонстрационного приложения.

```
package com.example;
import org.gnome.gtk.Gtk;
import org.gnome.gtk.Label;
import org.gnome.gtk.VBox;
import org.gnome.gtk.Window;
public class UselessGUIApp {
  private UselessGUIApp(){
    final Window mainWindow;
    mainWindow = new Window();
        mainWindow.setTitle("GUI Application Title");
    mainWindow.setSizeRequest(300, 100);
        VBox vbox = new VBox(false, 1);
    Label label = new Label("Label #1");
    vbox.add(label);
        mainWindow.add(vbox);
    mainWindow.connect(new Window.DeleteEvent() {
                        public boolean onDeleteEvent(Widget source, Event event) {
                                Gtk.mainQuit();
                                return false;
                        }
                });
    mainWindow.showAll();
  }
   public static void main(String[] args) {
    Gtk.init(args);
        new UselessGUIApp();
        Gtk.main();
  }
}
```

Ключевые вопросы:

1 Окно - основа основ любого GUI-приложения;

2 Свойства окон;

3 Классы VBox и HBox.

Литература:

1 Кузнецов И.Н. Научное исследование : методика проведения и оформления/ И. Н. Кузнецов . -3-е изд., перераб. и доп.. -М.: Дашков и К, 2008.-458 с

2 Рузавин Г.И. Методология научного познания : учеб. пособие : рек. УМЦ/ Г. И. Рузавин. - М.: ЮНИТИ-ДАНА, 2009.-288 с.

3 Кожухар В.М. Основы научных исследований: учеб. пособие/ В.М.Кожухар, -М.:Дашков и К, 2010. -216 с.

Практическая работа 4.

Model-View-Controller. Связь кода и элементов интерфейса. Примеры на C и Objective C. Элементы оконных интерфейсов.

Ключ к пониманию интерфейсов лежит в их сравнении с классами. Классы — это объекты, обладающие свойствами и методами, которые на эти свойства воздействуют. Хотя классы проявляют некоторые характеристики, связанные с поведением (методы), они представляют собой *предметы,* а не *действия,* присущие интерфейсам. Интерфейсы позволяют определять характеристики или возможности действий и применять их к классам независимо от иерархии последних. Допустим, у вас есть дистрибьюторское приложение, сущности которого можно упорядочить. Среди них могут быть классы *Customer, Supplier* и *Invoice.* Некоторые другие, скажем, *MaintenanceView* или *Document,* упорядочивать не надо. Как упорядочить только выбранные вами классы? Очевидный способ — создать базовый класс с именем типа *Serializable.* Но у этого подхода есть крупный минус. Одна ветвь наследования здесь не подходит, так как нам не требуется наследование всех особенностей поведения. C# не поддерживает множественное наследование, так что невозможно произвести данный класс от нескольких классов. А вот интерфейсы позволяют определять набор семантически связанных методов и свойств, способные реализовать избранные классы независимо от их иерархии.

Концептуально интерфейсы представляют собой связки между двумя в корне отличными частями кода. Иначе говоря, при наличии интерфейса и класса, определенного как реализующий данный интерфейс, клиентам класса дается гарантия, что у класса реализованы все методы, определенные в интерфейсе. Скоро вы это поймете на примерах.

Прочитав эту главу, вы поймете, почему интерфейсы являются такой важной частью C# в частности и программирования на основе компонентов вообще. Затем мы познакомимся с объявлением и реализацией интерфейсов в приложениях на C#. В завершение мы углубимся в специфику использования интерфейсов и преодоления врожденных проблем с множественным наследованием и конфликтами имен.

ПРИМЕЧАНИЕ Когда вы создаете интерфейс и в определении класса задаете его использование, говорят, что класс *реализует интерфейс,* или *наследует его.* Лично я считаю термин *"реализовать* " корректнее. Интерфейсы — это набор характеристик поведения, а класс определяется как реализующий их в противоположность наследования их от другого класса.

Применение интерфейсов

Чтобы понять, где интерфейсы приносят пользу, рассмотрим традиционную проблему программирования в Windows, когда нужно обеспечить универсальное взаимодействие двух совершенно различных фрагментов кода без использования интерфейсов. Представьте себе, что вы работаете на Microsoft и являетесь ведущим программистом команды по разработке панели управления. Вам надо предоставить универсальные средства, которые дают возможность клиентским апплетам "закрепляться" на панели управления, показывая при этом свой значок и позволяя клиентам выполнять их. Если учесть, что эта функциональность разрабатывалась до появления COM, возникает вопрос: как создать средства интеграции любых будущих приложений с панелью управления? Задуманное решение долгие годы было стандартной частью разработки Windows.

Как ведущий программист по разработке панели управления, вы создаете и документируете функцию (функции), которая должна быть реализована в клиентском приложении, и некоторые правила. В случае апплетов панели управления, Microsoft определила, что для их написания вам нужно создать динамически подключаемую библиотеку, которая реализует и экспортирует функцию *CPIApplet.* Вам также потребуется добавить к имени этой DLL расширение .cp! и поместить ее в папку Windows System32 (для Windows ME или Windows 98

это будет Win-do ws\System32, а для Windows 2000 — WINNT\System32). При загрузке панель управления загружает все DLL с расширением .cpl из папки System32 (с помощью функции *LoadLibrary),* а затем вызывает функцию *GetProcAddress* для загрузки функции *CPIApplet,* проверяя таким образом выполнение вами соответствующих правил и возможность корректного взаимодействия с панелью управления.

Как я уже говорил, эта стандартная модель программирования в Windows позволяет выйти из ситуации, когда вы хотите, чтобы ваш код универсальным образом взаимодействовал с кодом, который будет разработан в будущем. Однако это не самое элегантное в мире решение.

Главный недостаток этой методики в том, что она вынуждает включать в состав Клиента — в данном случае в код панели управления — большие порции проверяющего кода. Например, панель управления не может просто полагаться на допущение, что каждый .cpl-файл в папке является DLL Windows. Панель управления также должна проверить наличие в этой DLL функций коррекции и что эти функции делают именно то, что описано в документации. Здесь-то интерфейсы и вступают в дело. Интерфейсы позволяют создавать такие же средства, связывающие несовместимые фрагменты кода, но при этом они более объектно-ориентированны и гибки. Кроме того, поскольку интерфейсы являются частью языка C#, компилятор гарантирует, что если класс определен как реализующий данный интерфейс, то он выполняет именно те действия, о которых он заявляет, что должен их выполнять.

В C# интерфейс — понятие первостепенной важности, объявляющее ссылочный тип, который включает только объявления методов. Но что значит "понятие первостепенной важности"? Я хотел сказать, что эта встроенная функция является неотъемлемой частью языка. Иначе говоря, это не то, что было добавлено позже, после того как разработка языка была закончена. А давайте поподробнее познакомимся с интерфейсами, узнаем, что они собой представляют и как их объявлять.

ПРИМЕЧАНИЕ Разработчики на C++ могут воспринимать интерфейс в основном как абстрактный класс, в котором объявлены только чисто виртуальные методы в дополнение к другим типам членов классов C#, таким как свойства, события и индексаторы.

Объявление интерфейсов

Интерфейсы могут содержать методы, свойства, индексаторы и события, но ни одна из этих сущностей не реализуется в самом интерфейсе. Рассмотрим их применение на примере. Допустим, вы создаете для вашей компании редактор, содержащий элементы управления Windows. Вы пишете редактор и тестовые программы для проверки элементов управления, размещаемых пользователями на форме редактора. Остальная часть команды пишет элементы управления, которыми будет заполнена форма. Вам почти наверняка понадобятся средства проверки на уровне формы. В определенное время, скажем, когда пользователь явно приказывает форме проверить все элементы управления или при обработке формы, последняя циклически проверяет все прикрепленные к ней элементы управления, или, что более подходяще, заставляет элемент управления проверить самого себя.

Как предоставить такую возможность проверки элемента управления? Именно здесь проявляется превосходство интерфейсов. Вот пример простого интерфейса, единственным методом *Validate:*

```
interface IValidate {boo! ValidateO; }
```

Теперь вы можете задокументировать тот факт, что если элемент управления реализует интерфейс *IValidate,* то этот элемент управления может быть проверен.

Рассмотрим пару аспектов, связанных с приведенным кодом. Во-первых, вы не должны задавать для метода интерфейса модификатор доступа, такой *как public.* При указании модификатора доступа перед объявлением метода возникает ошибка периода компиляции. Дело в том, что все методы интерфейса открыты по умолчанию (разработчики на C++ могут также заметить, что, поскольку интерфейсы по определению — это абстрактные классы, не требуется явно объявлять метод как *"чисто" виртуальный* (pure virtual), прибавляя =0к его определению).

Кроме методов, интерфейсы могут определять свойства, индексаторы и события:

interface lExamplelnterface {// Пример объявления свойства.int testProperty { get; }// Пример объявления события, event testEvent Changed;// Пример объявления индексатора, string this[int index] { get; set; } }

Реализация интерфейсов

Поскольку интерфейс определяет связь между фрагментами кода, любой класс, реализующий интерфейс, *должен определять любой и каждый элемент этого интерфейса,* иначе код не будет компилироваться. Используя *IValidate* из нашего примера, клиентский класс должен реализовать лишь методы интерфейса. В следующем примере есть базовый класс *FancyControl* и интерфейс *IValidate.* Кроме того, в нем имеется класс

MyControl, производный от *FancyControl,* реализующий интерфейс *IValidate.* Обратите внимание на синтаксис и способ приведения объекта *MyControl* к интерфейсу *IValidate* для ссылки на его члены.

```
using System;
public class FancyControl
{
protected string Data; public string data {
get {
return this.Data; }
set {
this.Data = value; } } }
interface IValidate {
bool ValidateO; }
class MyControl : FancyControl, IValidate {
public MyControlQ
{
data = "my grid data";
}
public bool ValidateO {
Console.WriteLine("Validating...{0}", data); return true;
} >
class InterfaceApp {
public static void Main() {
MyControl myControl = new MyControK);
// Вызов функции для размещения элемента управления // в форме. Теперь для проверки элемента управления // редактор может выполнить такие действия:
IValidate val = (IValidate)myControl; bool success = val.ValidateO;
Console.WriteLineC'The validation of '{0}' was {1 Successful", myControl.data,
(true == success ? "" : "not ")); } }
```

С помощью определения такого класса и интерфейса редактор может запрашивать у элемента управления, реализует ли он интерфейс *IValidate* (ниже я покажу, как это сделать). Если это так, редактор может проверить этот элемент управления, а затем вызывать реализованные методы интерфейса. Возможно, вы спросите: "А почему бы мне просто не определить базовый класс для использования в этом редакторе, у которого есть чисто виртуальная функция*Validate!* Ведь после этого редактор будет принимать только элементы управления, производные от этого базового класса, да?"

Это решение повлечет суровые ограничения. Допустим, вы создаете собственные элементы управления, каждый из которых является производным от гипотетического базового класса. Как следствие, все они реализуют этот виртуальный метод *Validate.* Это будет работать, пока в один прекрасный день вы не найдете по-настоящему замечательный элемент управления и вам захочется использовать его в редакторе. Допустим, вы нашли компонент "сетка", написанный кем-то другим и поэтому не являющийся производным от нужного ре-

дактору базового класса "элемент управления". На C++ решение заключается в том, чтобы с помощью множественного наследования сделать сетку производной от компонента стороннего разработчика и базового класса редактора одновременно. Но C# не поддерживает множественное наследование.

Интерфейсы позволяют реализовать несколько характеристик поведения в одном классе. На C# можно создавать объекты, производные от одного класса, и в дополнение к этой унаследованной функциональности реализовать столько интерфейсов, сколько нужно для класса. Например, чтобы приложение-редактор проверяло содержимое элемента управления, связывало элемент управления с базой данных и последовательно направляло его содержимое на диск, объявите свой класс так:

public class MyGrid : ThirdPartyGrid, IValidate. ISerializable, IDataBound { }

Вопрос: "Как отдельный фрагмент кода узнает, реализован ли классом данный интерфейс?"

Запрос о реализации интерфейса с помощью is

В примере *InterfaceApp* вы видели код, использованный для приведения объекта *(MyControl)* к одному из реализованных в нем интерфейсов *(IValidate)* и затем для вызова одного из членов этого интерфейса *(Validate):*

MyControl myControl = new MyControlO;

IValidate val = (IValidate)myControl; bool success = val.ValidateO;

Что будет, если клиент попытается использовать класс так, как если бы в последнем был реализован метод, на самом деле в нем не реализованный? Следующий пример будет скомпилирован, поскольку интерфейс *ISerializable* является допустимым. И все же в период выполнения будет передано исключение *System.InvalidCastException,* так как в *MyGrid* не реализован интерфейс *ISerializable.* После этого выполнение приложения прервется, если только исключение не будет явно уловлено.

using System;

```
public class FancyControl
{
protected string Data; public string data {
get {
return this.Data; }
set {
this.Data = value;
> } }
interface ISerializable {
bool Save(); }
interface IValidate {
bool ValidateO; }
class MyControl : FancyControl, IValidate {
public MyControlO
{
data = "my grid data";
>
public bool ValidateO {
Console.WriteLine("Validating...{0}", data);
return true; > }
class IsOperatorlApp {
public static void Main()
{
MyControl rayControl = new MyControlO;
ISerializable ser = (ISerializable)myControl;
// Заметьте: в результате этого будет сгенерировано // исключение Sys-
```

```
tem.InvalidateCastException, поскольку // в классе не реализован интерфейс ISerializable. bool
success = ser.Save();
Console.WriteLine("The saving of '{0}' was {1}successful",
myControl.data,
(true == success ? "" : "not "));
} }
```

Конечно, улавливание исключения не повлияет на то, что предназначенный для выполнения код в этом случае не будет исполнен. Способ запроса объекта *перед* попыткой его приведения — вот что вам нужно. Один из способов — задействовать оператор *is.* Он позволяет в период выполнения проверять совместимость одного типа с другим. Оператор имеет следующий вид, где *выражение* — ссылочный тип:

выражение is *тип*

Результат оператора *is* — булево значение, которое затем можно использовать с условными операторами. В следующем примере я изменил код, чтобы проверять совместимость между классом *MyControl* и интерфейсом *ISerializable*перед попыткой вызова метода *ISerializable:*

```
using System;
public class FancyControl
{
protected string Data; public string data {
get
{
return this.Data;
}
set
{
this.Data = value;
} }
}
interface ISerializable {
bool Save(); >
interface IValidate {
bool ValidateO;
}
class MyControl : FancyControl, IValidate {
public MyControlO
{
data = "my grid data";
}
public bool ValidateQ {
Console.WriteLine("Validating...{0}", data);
return true; > }
class IsOperator2App {
public static void Main()
{
MyControl myControl = new MyControlO;
if (myControl is ISerializable) {
ISerializable ser = (ISerializable)myControl;
bool success = ser.SaveO;
Console. WriteLinef'The saving of '{0}' was "+ "{1}successful", myControl.data,
(true == success ? "" : "not ")); }
```

else {
Console.WriteLine("The ISerializable interface "+ "is not implemented."); > } }

Вы увидели, как оператор *is* позволяет проверить совместимость двух типов, чтобы гарантировать их корректное использование. А теперь рассмотрим его близкого родственника — оператор *as* и сравним их.

Ключевые вопросы:

1 Стандартная модель программирования в Windows;
2 Объявление интерфейсов;
3 Реализация интерфейсов.

Литература:

1 Кузнецов И.Н. Научное исследование : методика проведения и оформления/ И. Н. Кузнецов . -3-е изд., перераб. и доп.. -М.: Дашков и К, 2008.-458 с
2 Рузавин Г.И. Методология научного познания : учеб. пособие : рек. УМЦ/ Г. И. Рузавин. - М.: ЮНИТИ-ДАНА, 2009.-288 с.
3 Кожухар В.М. Основы научных исследований: учеб. пособие/ В.М.Кожухар, -М.:Дашков и К, 2010. -216 с.

Практическая работа 5.

Построение многооконных приложений. Отладка приложений. Использование моделей данных.

После запуска *C++ Builder* в вашем распоряжении имеется только одна форма *Form1* – стартовая форма будущего приложения для *Windows*. Если вы хотите построить приложение, использующее в своей работе, например, три окна – кликните мышью на значок New Form (Новая форма) два раза. Вы получили две новые формы: *Form2* и *Form3*. Такого же результата можно добиться по-другому. А именно в меню *File* выберите команду *New*, а затем Form. Во время работы приложения на экране могут одновременно отображаться все три формы, или любые две, либо любая одна из них. Для того чтобы из любой формы можно было управлять остальными формами, их файлы *Unit1.h*, *Unit2.h* и *Unit3.h* необходимо связать.

В заголовочной части файла *Unit1.cpp* для первой формы нужно дописать директивы:

#include "Unit2.h" //включить вторую форму в проект

#include "Unit3.h" //включить третью форму в проект

Расположите их под уже имеющейся директивой, которую сгенерировала сама среда разработки во время создания первой формы:

#include "Unit1.h" //включить первую форму в проект

Для второй и третьей форм сделать аналогичные записи в файлах *Unit2.cpp* и *Unit3.cpp*. После этого формы начнут «чувствовать» друг друга – станет возможно управление из любой формы оставшимися другими. Если в процессе разработки многооконного приложения какая-либо из форм окажется недоступной, для ее появления на экране выберите в меню *View* (Просмотр) команду *Forms…* (Формы) или с клавиатуры отработайте клавишный аккорд *Shift+F12*. Можно просто кликнуть на соответствующий значок с изображением нескольких форм.

На базе связанных между собой форм можно смело приступать к построению многооконного приложения. Для вызова из одной формы другой можно воспользоваться методом *Show()*, который загрузит указанную форму в оперативную память компьютера и покажет ее на экране. Метод *Hide()* скрывает форму от взора пользователя, без выгрузки ее из оперативной памяти. Например, строка программного кода, написанная в подходящей процедуре прерывания первой или второй формы, покажет на экране третью форму:

Form3->Show(); //показать третью форму

Убрать эту форму с экрана без выгрузки из оперативной памяти можно с помощью инструкции:

Form3->Hide(); //скрыть третью форму

Полностью закрыть форму можно с помощью инструкции:

Form3->Close(); //выгрузить третью форму

Построим многооконное приложение, состоящее из двух довольно самостоятельных частей. Первая часть будет информировать пользователя о текущих времени и дате, а также будет содержать календарь. Вторая часть предоставит в руки пользователя виртуальный генератор звуковых волн. Такой генератор будет интересен и полезен тем, кто увлекается физикой.

Создайте приложение на базе трех форм и свяжите их между собой. Общие размеры всех форм уменьшите примерно в два раза и расположите их пока произвольно в разных частях экрана. Измените заголовки каждой формы приложения. Пусть окна будут называться: «Главное меню», «Электронные часы. Календарь» и «Виртуальный генератор звуковых волн». Снабдите первую и вторую форму подходящими фоновыми рисунками с помощью знакомого уже вам компонента *Image*.

На первой форме установите две электронные кнопки. Увеличьте их ширину в два-три раза, для того чтобы их можно было подписать «Электронные часы» и «Виртуальный генератор» соответственно. Напишите функцию обработки *Button1Click*:

Form2->Show(); //показать вторую форму

Form1->Hide(); //скрыть первую форму

Затем для второй кнопки напишите функцию обработки *Button2Click*:

Form3->Show(); //показать третью форму

Form1->Hide(); //скрыть первую форму

На второй форме установите электронную кнопку для возврата в главное меню и снабдите ее надписью «Назад». Напишите для нее функцию обработки *Button1Click*:

Form1->Show(); //показать первую форму

Form2->Hide(); //скрыть вторую форму

Обустройте третье окно аналогично второму, соответственно изменив программный код. У второй и третьей форм удалите три кнопки управления окном. Для этого обратитесь к их составному свойству *BorderIcons*, которое отвечает за вид окна, кликнув на «плюс». Чтобы исчезли кнопки управления окном, свойству *biSystemMenu* (Системное меню) придайте значение *false*.

Запустите приложение на исполнение. Посредством электронных кнопок попробуйте управлять многооконным приложением для *Windows*. Если все окна корректно появляются и исчезают, то можно достроить второе и третье окна, так чтобы они отвечали задуманному нами в самом начале назначению.

На вторую форму добавьте компоненты *Label1* и *Label2*. Сюда же установите компонент *Timer1* из вкладки *System* и компонент *MonthCalendar1* (Ежемесячный календарь) из вкладки *Win32*. Для компонента *Timer1* его свойству *Interval* придайте значение 10. В функцию обработки *Timer1Timer* запишите инструкции:

Label1->Caption = Date(); //показать дату

Label2->Caption = Time(); //показать время

Через каждую одну сотую секунды (10 миллисекунд) дата и время будут обновляться. Если вы хотите чтобы дата и время одновременно выводились в одном поле вывода текста *Label1*, то две предыдущие инструкции можно заменить одной:

Label1->Caption = Now(); //показать дату и время

Если вам необходимо чтобы календарь показывал на экране несколько месяцев одновременно, значительно увеличьте размеры компонента *MonthCalendar1*. При этом размеры второй формы можно и не увеличивать, так как если ежемесячный календарь не сможет разместиться на форме по размерам, автоматически появятся полосы прокрутки по вертикали и горизонтали.

После проверки функций второго окна, можно приступать к построению третьего окна, которое будет представлять виртуальный генератор звуковых волн.

На третью форму добавьте компоненты *Button2*, *Label1*, *TrackBar1*, *Image1*, *Image2*. Компонент *TrackBar1* (Движок-регулятор) можно извлечь из вкладки *Win32*. Ширину ком-

понента *TrackBar1* увеличьте примерно в два-три раза. Свойству *Max* (Максимальное положение движка) этого компонента придайте значение 50. Очистите свойство *Caption* у компонента *Label1*. В функцию обработки *Button2Click* впишите инструкцию, которая будет генерировать колебания звуковой частоты:

Beep(TrackBar1->Position*100, 1000); //звук

Этот оператор, отвечающий за генерацию звука, имеет два аргумента. Первый задает частоту звука в герцах, второй – длительность в миллисекундах. Свойство *Position* (Позиция движка) компонента *TrackBar1* управляет изменением частоты звука. В нашем случае можно получить звук частотой от 100 до 5000 Гц с шагом в 100 Гц длительностью в одну секунду. Электронную кнопку *Button2Click* назовите «Пуск». Генератор уже работоспособен, но для вывода информации о частоте звука необходимо в функцию обработки *TrackBar1Change* (Изменение позиции движка) записать инструкцию:

Label1->Caption = "Частота звука = " +

IntToStr(TrackBar1->Position)*100 + " Гц";

Заметьте, здесь одна инструкция записана с переносом, а поэтому расположилась в двух строках.

Запустите приложение и проверьте работоспособность третьего окна. Поэкспериментируйте с оператором звука *Beep*, изменяя оба его аргумента. Будьте осторожны с величиной второго аргумента, так как он отвечает за время звучания.

Добавьте в окно генератора простейшую анимацию. Для этого вставьте два подходящих изображения, которые будут отображать состояние генератора, в поля компонентов *Image1*, *Image2*. Этими изображениями могут быть, например, обезьяна в двух разных состояниях. Когда генератор молчит – обезьяна в задумчивости, когда присутствует звук – обезьяна улыбается разводит руки. Оба изображения наложите, друг на друга, так как появляться они будут в разное время. Свойству *Visible* компонента *Image2* придайте значение *false*. Теперь функция обработки *Button2Click* будет выглядеть так:

Beep(TrackBar1->Position*100, 1000); //звук

Image1->Visible = true; //показать первый рисунок

Image2->Visible = false; //скрыть второй рисунок

Необходимо так же будет написать функцию обработки Button2MouseDown:

Image1->Visible = false; //скрыть первый рисунок

Image2->Visible = true; //показать второй рисунок

Запустите приложение. Посмотрите, как теперь работает виртуальный генератор в сопровождении анимации.

Самостоятельно достройте третье окно, так чтобы можно было регулировать длительность звучания, например, до трех секунд с шагом в полсекунды. Для регулировки длительности звука воспользуйтесь компонентом *TrackBar2*. Для вывода информации о длительности звукового сигнала используйте компонент *Label2*.

Ключевые вопросы:

1 Построение многооконных приложений;

2 Отладка приложений;

3 Использование моделей данных.

Литература:

1 Кузнецов И.Н. Научное исследование : методика проведения и оформления/ И. Н. Кузнецов . -3-е изд., перераб. и доп.. -М.: Дашков и К, 2008.-458 с

2 Рузавин Г.И. Методология научного познания : учеб. пособие : рек. УМЦ/ Г. И. Рузавин. -М.: ЮНИТИ-ДАНА, 2009.-288 с.

3 Кожухар В.М. Основы научных исследований: учеб. пособие/ В.М.Кожухар, -М.:Дашков и К, 2010. -216 с.

Практическая работа 6.

Особенности построения приложений для WEB. Использование элементов GUI. CSS, HTML, Javascript, Ajax, Apache.

Со времен появления Internet Web-разработчиками были написаны миллионы байт HTML-кода. Для создания Web-сайтов и Web-приложений использовались и продолжают использоваться три основные средства разработки. Это непосредственно теги HTML, какой-либо интерпретируемый (скриптовый) язык (JScript, JavaScript, VBScript и т. п., везде далее — JavaScript, который мы будем использовать в рабочих примерах, а исключения оговаривать отдельно) и каскадные стилевые таблицы (CSS). Помимо этих основных средств разработки в содержимое Web-приложения могут входить изображения различного формата, анимация и т. д., которые, в общем, компонентами HTML не являются.

Давайте рассмотрим Web-интерфейс с точки, зрения HTML или, точнее, DHTML. Если рассмотреть код, из которого состоит не самым примитивным образом оформленная Web-страница, можно увидеть, что теги HTML отвечают за то, какие элементы должны присутствовать на странице, CSS — за отображение этих элементов и, наконец, код JavaScript — за все изменения, которые могут происходить с этими элементами. Таким образом, если говорить о создании Web-интерфейса, то можно сказать, что его видимую составляющую представляют теги HTML и конструкции CSS, а невидимую — операторы языка JavaScript.

Такая система разработки Web-приложения обладает некоторыми недостатками. Давайте вернемся к примеру со служащим отдела кадров и попытаемся представить себе Web-интерфейс приложения, которое позволяет ему обрабатывать регистрационные карточки и резюме устраивающихся на работу. Web-приложение, позволяющее только лишь обработать поля форм и редактировать текст, само по себе не будет слишком сложной программой или, что точнее сказать, страницей.

Но если принять во внимание то, что служащему необходимо создавать новые документы (например, заводить карточки), удалять старые, просматривать список уже имеющихся документов, то сразу видно, что функциональность такого Web-приложения становится гораздо выше. Оно уже должно содержать меню, каждый пункт которого сопоставлен с командами удаления, добавления и т. д. документов; отдельное окно со списком документов, а на практике, чаще всего, с деревом, каждая вершина которого представляет собой либо документ, либо папку с документами.

Представьте себе коллектив разработчиков, который пытается создать Web-интерфейс для этого приложения или для любой другой, столь же комплексной задачи. При этом они используют HTML, CSS и JavaScript, под совокупностью которых мы и будем подразумевать DHTML.

Во-первых, разработчики-программисты в связи с большим объемом кода JavaScript будут вынуждены написать некоторое количество вспомогательных функций, которые, скорее всего, будут содержать часто повторяющийся код. Затем они должны будут включить этот код (непосредственно или путем подключения дополнительного файла) в каждую страницу приложения, которая содержит хотя бы один элемент управления. Очевидно, что любая из таких страниц не станет от этого загружаться быстрее. А если на странице находится форма, гиперссылка или что-либо еще, заставляющее эту страницу перерисовываться при активировании этого объекта пользователем?

Поэтому можно сделать вывод, что малая скорость загрузки и работы Web-приложения, созданного исключительно средствами DHMTL, и большой его объем являются главными недостатками. То же самое касается и конструкций CSS или форматирующих тегов (это зависит от предпочтений разработчиков), описывающих стили оформления наиболее часто встречающихся элементов, например, пунктов меню.

Во-вторых, вспомним, что разработчиками являются не только программисты, но и дизайнеры, люди, имеющие к программированию несколько отдаленное отношение. Им для того, чтобы встроить в интерфейс, допустим, статичное или анимированное изображение, нужно обращаться к Web-мастерам и программистам, у которых, между нами говоря, свои

проблемы. Если дело касается того, чтобы просто вставить рисунок, то это — еще полбеды. Представляете выражение лица программиста, которого в двадцать четвертый раз просят сдвинуть рисунок, скажем, на два миллиметра вправо, да еще непонятно с какой целью? Отсюда следует, что при таких средствах разработки у коллектива полностью отсутствует разделение труда, т. е. каждый из них должен помимо своей профессии иметь представление о многих вещах, с ней не связанных.

В-третьих, отметим, что существующие на сегодняшний день средства отладки DHTML весьма бедны. А средства автоматического или визуального построения элементов управления отсутствуют полностью. Пусть даже у разработчиков не возникает головной боли по поводу управления памятью или приведения типов, но что если в громоздком HTML-документе нужно найти причину не совсем корректного отображения какой-то его части? Таким образом, третьим недостатком DHTML является его поначалу кажущаяся достоинством вольность интерпретации. Ведь ни один HTML-документ сам по себе не откажется загружаться и не выдаст сообщение об ошибке, если, например, какой-либо его открывающий тег (блочный или встроенный — не важно) не имеет закрывающего.

Преимущества XML

Как вы уже, наверное, догадываетесь, язык XML в полной мере компенсирует все эти недостатки. Так как пример со служащим отдела кадров не является всеохватывающим, на самом деле таких недостатков гораздо больше. Если же ограничиться рамками этого примера, то, задействовав XML, получим следующие решения вышеперечисленных проблем.

Обычно XML-документ содержит не только какую-либо структурированную информацию, но и правила ее трансформации, т. е. представляет собой, как минимум, два файла. Один из этих файлов содержит данные, структурированные с помощью тегов, названия которых1 придуманы создателем документа. Второй — правила интерпретации этих тегов. Первый файл обычно имеет расширение xml, а второй — xsl. Такое свойство XML-документов называется разделением данных и их представления.

Программист-разработчик может извлечь из этой ситуации несколько полезных усовершенствований для своего способа программирования. Во-первых, если возникает необходимость в частом использовании какой-либо громоздкой HTML-конструкции (например, элемента текста, или рисунка, или и того и другого вместе с множеством правил форматирования), то при использовании XML достаточно создать один тег или группу тегов, единственный раз задать правила их форматирования и далее вместо всей конструкции использовать только эти созданные теги. Для разработчика такая модульность кода продукта может оказаться немаловажной.

Рассмотрим следующий код:

```
<card number="111">
  <surname> Иванов </surname>
  <name> Сергей </name>
  <telephone> 01-01-01 </telephone>
...
</card>
```

Оперировать тегами с такими именами как card, name и т. д. гораздо проще и понятнее, чем тегами языка HTML. И, кроме того, каждый из этих тегов может представлять собой набор правил форматирования любой степени сложности. Это не означает, что нужно совсем отказаться от HTML, ведь, говоря о правилах форматирования, мы подразумеваем использование именно этого языка. Но, с другой стороны, если работать только на HTML, то результатом будет замена каждого тега из вышеприведенного примера на его HTML-интерпретацию. Таким образом, объем кода для каждой карточки резко возрастет, а это, согласитесь, не очень удобно.

Во-вторых, для дизайнеров такое положение вещей также приносит некоторую, хотя и весьма небольшую, выгоду. Мы говорили, что разработчики приложений или сайтов на языке HTML, программисты или дизайнеры должны владеть не только своей профессией, но и

смежными специальностями. Откровенно говоря, XML не решает эту проблему и не собирается этого делать в будущем. По крайней мере, от разработчиков стандартов данного языка не поступает никаких внятных предложений по этому поводу.

Единственное, хотя и очень небольшое преимущество XML состоит в том, что XML-документ состоит не из одного файла, а из нескольких, и предполагается, что можно организовать работу программистов и дизайнеров так, чтобы каждый работал над своей группой файлов. Но на практике их деятельность неизбежно пересекается, и поэтому определенно можно сказать только то, что XML решает проблему разделения труда лучше, чем HTML.

Что касается средств отладки XML-кода, то, как видно из примера, каждая единица информации заключена в логические скобки — открывающий и закрывающий теги. В языке XML закрывающий тег необходим всегда, как и кавычки, в которые заключены значения атрибутов. XML изначально задумывался как язык разметки, не содержащий двусмысленностей интерпретации, и это отражено в соответствующих стандартах и спецификациях. Такая жесткость правил синтаксиса языка позволяет значительно сократить количество ошибок в документе, а также отыскать ту часть кода, которая отвечает за несоответствующим образом отображающуюся часть документа. При несоблюдении правил документ просто не будет отображаться в окне браузера, вместо этого появится сообщение об ошибке.

Мы рассмотрели только некоторые недостатки HTML и то, как XML справляется с ними. Но существует ряд задач, для которых HTML абсолютно не предназначен, и это нельзя назвать его недостатком. Основной функцией этого языка является только лишь отображение информации, в то время как язык XML охватывает ее смысл и структуру. В этом и состоит революционность XML и суть этой технологии.

Сопоставление XML и HTML

Как нам удалось показать, XML не может выступать отдельно от HTML в том случае, когда информация, содержащаяся в XML-документе, нуждается в некотором представлении, или, проще говоря, в отображении (точнее говоря, может, но его совместное использование с HTML несколько "уютнее"). Теперь следует обозначить границу между этими двумя языками. Мы уже можем сделать вывод, что язык XML никогда не вытеснит HTML, но и не оставит его неизменным. Влияние XML на HTML выражается в появлении такого стандарта как XHTML, который представляет собой, грубо говоря, HTML, дополненный жесткими правилами разметки языка XML. И если между XML и HTML не может быть конкурентной борьбы в силу различия их функций, то эта борьба возможна и уже происходит между HTML и XHTML.

Для того чтобы наилучшим образом проиллюстрировать грань между XML и HTML, вспомним два типа текстовых редакторов. Первый тип представляет собой редакторы, работающие по принципу WYSIWYG (What You See Is What You Get — что ты видишь, то ты и получишь). Согласно данному принципу, вид текста при вводе будет идентичен отображению на экране или распечатке на принтере. Примером такого редактора может служить Notepad. Примером реализации второго типа редакторов (когда вводимый и отображаемый документы не совпадают) могут быть редактор TeX или, скажем, SGML+DSSSL. Существование этих двух типов редакторов говорит о том, что способ хранения информации может отличаться от способа ее отображения.

Учитывая требования к Web-приложению, отметим, что информация, находящаяся на Web-сервере, должна храниться в наиболее компактной форме. И в то же время она, если это потребуется, должна выглядеть по-разному для разных пользователей. В примере со служащим отдела кадров мы говорили, что к регистрационной карточке устраивающегося на работу могут иметь доступ пользователи, выполняющие разные функции. Поэтому кому-то из них могут понадобиться Паспортные данные, другому — виды деятельности на прежних местах работы и т. д. Таким образом, разные пользователи, обращаясь к регистрационной карточке, могут увидеть разные данные об устраивающемся на работу. А может быть и те же самые, но по-разному отформатированные.

Если мы говорим о текстовом документе, с которым работает только один человек, то для его обработки наиболее подходит редактор, работающий по принципу WYSIWYG. Но если речь заходит о множественном доступе к документу, т. е. о многопользовательском приложении, то информация должна быть либо продублирована соответствующим образом для каждого пользователя (что не удовлетворяет требованию компактности), либо храниться в некоем универсальном виде, отличном от того, в котором отображается.

Если глубже проанализировать различия двух типов текстовых редакторов, то можно заметить, что те из них, которые относятся к типу WYSIWYG, обрабатывают весь текст документа равноправно, т. е. вне зависимости от того, каким смысловым содержанием наделен этот текст (паспортные ли это данные или что-либо еще). Второй тип редакторов, хоть и на примитивном уровне в случае нашего примера, уделяет некоторое внимание структуре данных, содержащихся в документе.

Впрочем, текстовые редакторы — это всего лишь аналогия, но, тем не менее, она позволяет отделить друг от друга HTML и XML. Если первый занимается только представлением информации, то второй — еще и ее хранением в некоторой специфической форме. Таким образом, можно вывести формулу: XML = структурированная информация + способ(ы) ее представления, где роль последних отводится языку HTML.

Проект XML: прорисовка формы

Итак, что же такое язык XML и чем он не является?

Несмотря на то, что в названии (расширяемый язык разметки) слово "разметка" присутствует, XML не содержит ни одного правила или набора символов, который можно было бы назвать "разметкой". Вообще говоря, разметка представляет собой описание логической структуры документа и предназначена для определения частей документа и введения над ними отношений порядка. Например, в HTML тег <p> означает, что заключенный в него текст является параграфом, в TeX команда /frac{...}{...} — о том, что текст в первых фигурных скобках является числителем, а во вторых — знаменателем дроби. Поэтому не совсем верно называть XML языком разметки. Он представляет собой скорее инструментарий для создания такого языка, который необходим для разметки обрабатываемого класса документов. Примерами таких классов могут быть математические или химические публикации, гипертекст и обычный текст и все, что угодно.

Для великого множества типов документов, будь то газетные статьи, книги, научные публикации или различные формуляры, и был создан язык XML — универсальный способ для описания методов хранения и управления информацией любого вида, единственным ограничением на которую является возможность ее вербализации. Это описание представляет собой, фактически, моделирование некоего исходного, неформализованного текста.

Модель XML-документа строится в зависимости от того, какую смысловую нагрузку несут части текста и как они взаимосвязаны друг с другом. Типы связей при этом могут быть самыми разными. Например, в случае книги, такой связью может быть относительное расположение названий или нумерации частей, глав и содержащегося в них текста. Или это может быть связь по типу гиперссылки, когда документ непосредственно не содержит какую-то свою часть, а только ссылается на нее.

Модель или логическая структура XML-документа служит только одной цели, состоящей в том, чтобы иметь возможность работать с этим документом, как с набором динамических (изменяемых во времени, а не формируемых по требованию) данных, а не просто как со статическим текстом, который можно только просматривать. При наличии жесткой структуры и строгой однозначности синтаксиса, в XML-документе очень просто найти и получить доступ к любой содержащейся в нем единице информации. Работая, например, с таблицей и используя механизм адресации XPath, можно легко получить и обработать содержимое произвольной ее ячейки. В то время как средствами HTML то же самое действие осуществить значительно труднее, да, в общем-то, и не нужно. Язык HTML послужит в данном случае только для того, чтобы эту таблицу отобразить.

Как мы уже говорили, XML-документ состоит из двух частей. Причем вторая часть, не являясь обязательной, служит средством отображения или публикации того, что содержится в первой. Первая же часть и содержит сам XML-код, т. е. текст и его разметку, созданную пользователем. Оперируя такими инструментами, как адресация любого элемента или его атрибута, создание "на лету" любой конструкции XML, а также всеми возможностями языка запросов к XML-документу, мы превращаем его в базу данных с более сложной, но четко определенной структурой.

Здесь под языком запросов мы понимаем возможность чтения и, самое главное, записи содержимого элемента или его атрибута, а также удаления элемента, атрибута или их содержимого, причем все эти операции проводятся над внешним документом. Хотя нужно отметить, что на сегодняшний день не существует удовлетворительного языка запросов, осуществляющего запись и удаление в XML-документе. Большинство из распространяемых сегодня языков стремятся к тому, чтобы производить сложные многокритериальные выборки из одного или нескольких XML-документов, при этом сохраняя простоту и гибкость запроса.

Несмотря на широкий спектр возможностей языка XML, следует понимать, что он не стремится заменить собой ни базы данных, ни существующие средства создания Web-приложений, ни даже средства публикаций. Этот язык не поглощает функций ни одной из этих технологий, но объединяет некоторые из них. И, в то же время, он сам является отдельной технологией хранения и публикации данных, а также служит для организации взаимодействия некоторых других технологий и направлений, как правило, имеющих отношение к Internet. Для того чтобы говорить о роли, которую играет XML в виде одной серверной технологии среди группы других, нужно сначала определить предпосылки его возникновения и проследить путь его эволюционного развития.

W3-консорциум и X-технологии

Временем рождения XML можно считать 1996 год, в конце которого появился черновой вариант спецификации языка, или 1998, когда эта спецификация была утверждена. Разумеется, для понимания проблем современности необходимо знание истории, но, говоря об истории XML, предоставим каменный век историкам и начнем сразу с 1986 года — года появления языка SGML.

SGML (Standard Generalized Markup Language — Стандартный Обобщенный Язык Разметки) заявил о себе как гибкий, комплексный и всеохватывающий мета-язык для создания языков разметки. И хотя понятие гипертекста появилось в 1945 году, этот язык не имеет гипертекстовой модели. Создание SGML можно с уверенностью назвать попыткой, объять необъятное, т. к. он объединяет в себе такие возможности, которые крайне редко используются все вместе. В этом и состоит его главный недостаток — сложность и, как следствие, дороговизна этого языка ограничивает его использование только крупными компаниями, которые могут позволить себе купить соответствующее программное обеспечение и нанять высокооплачиваемых специалистов. Кроме того, у небольших компании редко возникают настолько сложные задачи, чтобы привлекать к их решению SGML.

Наиболее широко SGML применяется для создания других языков разметки, именно с его помощью был создан язык разметки гипертекстовых документов — HTML, спецификация которого была утверждена в 1992 году. Его появление было связано с необходимостью организации стремительно увеличивающегося массива документов в сети Internet. Бурный рост количества подключений к Internet и, соответственно, Web-серверов повлек за собой такую потребность в кодировке электронных документов, с которой не мог справиться SGML вследствие высокой трудности освоения. Появление HTML — очень простого языка разметки — быстро решило эту проблему, легкость в изучении и богатство средств оформления документов сделали его самым популярным языком для пользователей Internet.

Но, по мере роста количества и изменения качества документов в Сети, росли и предъявляемые к ним требования, и простота HTML превратилась в его главный недостаток. Ограниченность количества тегов и полное безразличие к структуре документа побудили разработчиков в лице консорциума W3C к созданию такого языка разметки, который был бы не

столь сложен, как SGML, и не настолько примитивен, как HTML. В результате, сочетая в себе простоту HTML, логику разметки SGML и удовлетворяя требованиям Internet, появился на свет язык XML. И несмотря на молодость (всего-то 4 года), уже успел превратиться в целый набор технологий, призванных выполнять самые разные действия по обработке информации как в сети Internet, так и вне ее.

Посетив сайт консорциума W3C (www.w3.org), вы можете познакомиться с несколькими десятками этих, так называемых, X-технологий. Здесь мы перечислим и кратко опишем некоторые из них, в том числе и, собственно, XML, которому посвящена книга.

XML

Расширяемый язык разметки (extensible Markup Language, XML) описывает класс объектов XML document, а также частично работу компьютерных программ, обрабатывающих объекты с данными, реализующими этот класс. XML — это прикладной уровень или усеченная форма SGML, Стандартного Обобщенного языка разметки [ISO 8879]. По своему построению, XML-документ является полноценным SGML-документом.

XSL

Расширяемый язык стилевых таблиц (extensible Stylesheet Language, XSL) состоит из двух частей: языка для трансформации XML-документов и из XML-словаря, определяющего семантику форматирования. Стилевая таблица XSL определяет представление класса XML-документов, описывая, как представитель класса, трансформируемый в XML-документ, использует словарь форматирования.

XSLT

Язык XSLT (XSL Transformations) разработан как часть XSL. XSL определяет стилизацию XML-документа, задействуй XSLT для объяснения того, как документ трансформируется в другой XML-документ на основе словаря форматирования. Кроме того, XSLT может использоваться независимо от XSL. Однако XSLT не является единым комплексным языком трансформации XML-документов. Он, скорее, предназначен для тех видов трансформаций, которые необходимы, когда XSLT используется как часть XSL.

XPath

Главной задачей XPath является адресация частей XML-документа. Для этой цели XPath представляет XML-документ как некоторое дерево узлов различного типа. Поэтому помимо адресации в XPath обеспечиваются минимальные возможности по обработке данных различных типов. Третьей задачей XPath можно назвать динамическое генерирование контента, если этот контент не может быть создан до первого обращения к документу.

XSL Formatting Objects

XSL FO (Formatting Objects — форматирующие объекты) — это набор классов, в терминах которых отражена семантика форматирования. Они представляют собой узлы дерева, полученного в результате XSL-трансформации. Классы форматирующих объектов определяют такие полиграфические термины, как страница, параграф и т. п. Лучший контроль над представлением этих объектов обеспечивается множеством свойств форматирования. Это такие свойства, как отступы, тени, промежутки между словами и буквами и т. п. В XSL классы свойств и объектов форматирования обеспечивают словарь для отражения цели представления.

XHTML

Расширяемый HTML (extensible HTML, XHTML) является результатом применения правил синтаксиса XML к стандарту HTML. Расширяемость XHTML заключается в возможности определения таких конструкций, как элементы, атрибуты, сущности и т. п. с помощью подключения определения типа документа (Document Type Definition, DTD).

RDF

Инструментарий описаний ресурсов (Resource Description Framework, RDF) — это набор инструментов для работы с метаданными. Он обеспечивает единую, стандартизированную среду управления внутренним (без вмешательства человека) взаимодействием приложений, которые обмениваются в Web информацией, понимаемой машинами. RDF делает ударение

на легкость автоматизированной обработки Web-ресурсов. Метаданные RDF могут быть использованы в самых разных областях работы приложений. В общем, RDF обеспечивает основу для элементарных инструментов авторизации, поиска и редактирования данных, понимаемых машинами, в Web, содействуя, таким образом, трансформации Web в аппаратно обрабатываемое хранилище информации.

XML Schema

Язык XML Schema используется для объявления элементов и атрибутов в XML-документе с целью его структуризации. Кроме того, этот язык предоставляет расширяемые возможности для определения типов данных элементов и атрибутов.

XBase

XML Base представляет собой аналог элемента BASE из языка HTML. Он описывает механизм предоставления сервисов базовых URI (Uniform Resource Locator, универсальное местоположение ресурса) для XLink.

XLink

Расширяемый язык связывания (XML Linking Language, XLink) определяет конструкции, которые могут быть вставлены в XML-документы для описания связей между объектами. Он использует синтаксис языка XML для создания структур, которые могут служить как для описания простых однонаправленных гиперссылок, определяемых в HTML, так и для более сложных связей.

XPointer

Расширяемый язык указателей (XML Pointer Language, XPointer) — язык, разработанный для совместного использования с XLink. XPointer определяет конструкции, которые поддерживают адресацию во внутренних структурах XML-документов. В частности, он предназначен для специфических ссылок на элементы, символьные строки и другие части XML-документов.

Xlnclude

Расширяемый язык встраивания (XML Include Language, Xlnclude) предназначен для объединения блоков информации в формате XML (XML infosets) в единый составной информационный блок. Спецификация документов XML (или информационных блоков), которые должны быть объединены, а также процесс объединения описываются с помощью дружественного языку XML синтаксиса (элементов, атрибутов, ссылок на URI).

XQL

Расширяемый язык запросов (XML Query Language, XQL) — это нотация для адресации и фильтрации элементов и текста в XML-документах. XQL является естественным расширением схемы (pattern) синтаксиса XSL. Он обеспечивает выразительную и простую нотацию для указания (pointing) на специфические узлы, а также для поиска узлов со специальными, частными характеристиками. Этот язык, базируясь на возможностях XSL, обеспечивает идентификацию классов узлов путем добавления булевой логики, фильтров, индексации в коллекциях узлов и т. д.

SOAP

Простой протокол доступа к объектам (Simple Object Access Protocol, SOAP) представляет собой облегченный протокол для обмена информации в децентрализованной, распределенной среде. Этот базирующийся на XML протокол состоит из трех частей:

- конверт, который служит для определения содержимого сообщения и способов его обработки;
- набор правил кодировки (encoding rules), предназначенных для описания определенных в приложении типов данных;
- соглашение о вызовах удаленных процедур (Remote Procedure Call, RFC) и об их результатах.

В SOAP заложены потенциальные возможности взаимодействия с другими протоколами, такими как HTTP и его расширенным инструментарием (HTTP Extension Framework).

MathML

Математический язык разметки (Mathematical Markup Language, MathML) — базирующийся на XML язык описания математических формул. Представляет собой расширение XML, предназначенное как для отображения, так и для обработки структуры математических публикаций. Этот язык предназначен для создания инструментария разметки математики в Web.

CML

Химический язык разметки (Chemical Markup Language, CML) является аналогом MathML для публикаций в Web статей по химической тематике. На самом деле, Internet-технологий, разными способами обрабатывающих данные в Сети и для Сети, гораздо больше. Мы не сильно ошибемся, если всю их совокупность будем обозначать одним термином — XML.

XML среди других серверных технологий

Несмотря на многообразие возможностей, которые предоставляет язык XML, было бы заблуждением думать, что программное обеспечение, необходимое для создания Web-приложения, состоит только из синтаксического анализатора (парсера) XML. Большинство функций этого языка направлено на обработку данных, содержащихся внутри документа, но как обстоит дело с операциями над самими документами? Кроме того, информация, содержащаяся в документе, может быть структурирована различными способами, в зависимости от специфики задачи. Зададимся вопросом, для работы с какими типами таких структур подходит язык XML, а для каких нет?

Мы уже определили в общих чертах, из чего состоит и каким требованиям должно удовлетворять Web-приложение. Не будем здесь развивать эту тему более подробно, просто рассмотрим совокупность функций Web-приложения и языка XML. Для этого предположим, что рассматриваемое нами Web-приложение достаточно компактно, пусть оно обслуживает не более чем несколько сотен пользователей, и в задачи этих пользователей входит только лишь работа с документами. Пусть также наше приложение не производит ни научных расчетов, ни громоздких операций с документами в режиме реального времени. И вместе с тем, оно успешно справляется с поставленными перед ним задачами и делает это с удовлетворительной скоростью.

Прежде всего, заметим, что если мы решили воспользоваться помощью XML для построения Web-приложения, то это еще не значит, что нам необходимо отказаться от программного обеспечения, служащего для представления HTML-документов. Ведь некоторые документы, обрабатываемые приложением, могут служить только для представления информации. Единственная операция, которую может осуществить над ними пользователь — просмотр, поэтому логическая структура таких документов не имеет никакого значения.

Следовательно, для того, чтобы приложение имело возможность обрабатывать статические документы, одним из его компонентов должен быть Web-сервер. Можно, разумеется, перевести все HTML-документы в формат XML, но тогда логично будет заключить, что таким образом Web-приложение будет работать медленнее, т. к. ему понадобится время на преобразование XML в HTML. Хотя нужно признаться, что на практике такое замедление работы случается не всегда.

Основная функция языка XML, как мы уже говорили, состоит в структуризации информации, т. е. создавая XML-документ, мы фактически строим формализованную модель данных. Если при этом количество этих данных слишком велико для того, чтобы они образовывали один документ, то возникает множество таких документов, средствами коммуникации между которыми служат языки запросов.

Даже в том случае, когда количество документов составляет несколько тысяч или десятков тысяч, совокупность их описаний также представляет собой набор данных, который может быть структурирован. Другими словами, можно создать один или несколько XML-документов, которые бы описывали все множество документов, хранящихся на сервере. В

них можно включить информацию об уровнях доступа пользователей этого сервера, простые или сложные индексы документов и даже некоторый механизм обработки транзакций.

Но проделав всю эту работу, мы всего-навсего изобретем велосипед под названием "служба каталога". В этом нет никакой необходимости, т. к. на данный момент существует достаточное количество соответствующих серверов на любой вкус как коммерческих, так и бесплатных. Единственным ис ключением является ситуация, когда количество документов слишком мало для привлечения услуг этой службы.

Так как Web-приложение помимо статических данных обрабатывает еще и динамические, то должен существовать механизм, позволяющий обновлять связи между документами. Это означает, что при изменении какой-либо информации о документе или содержащейся в документе, на которую ссылается другой документ, в этом другом документе также должны произойти соответствующие изменения. Например, создание одного XML-документа может повлечь за собой обновление общего списка этих документов, т. е. другого XML-документа. Другим примером может быть генерирование отчетов — документов, содержащих сводную информацию из других источников.

Опять же, язык XML может быть привлечен для решения этой задачи, но когда речь заходит о каких-либо действиях произвольного характера (то есть не только представление и работа со структурой), то лучше воспользоваться каким-либо языком программирования. Ведь такая простая операция как получение текущей даты не может быть выполнена средствами языка разметки, каким бы расширяемым он не был.

В самом начале мы договорились использовать Java в качестве языка программирования, который, как и XML, представляет собой платформу для создания различных Internet-технологий. Одной из таких технологий является выполненный средствами Java инструмент под названием "сервлет" (servlet), который грубо можно определить как "атомарный" код, выполняющийся на сервере и возвращающий клиенту результаты своего выполнения. Причем клиент и сервер не обязательно должны находиться на различных машинах и взаимодействовать по сети.

Благодаря механизму сервлетов, мы можем объединить функциональность языка XML в плане обработки документов с мощной логикой языка программирования. Подробнее о взаимодействии XML, Java и сервлетов см. главы 6 и 8.

Как XML-документы, так и сервлеты представляют собой множество объектов, которое может быть упорядочено. Но поскольку сервлеты представляют собой наборы действий, т. е. программы, и в свою очередь являются приложениями, то для их обслуживания необходим отдельный сервер, который так и называется — сервер приложений (application server). Вообще в задачу сервера приложений входит не только обработка сервлетов, но и любых других приложений, взаимодействующих с сервером и друг с другом по некоторому единому протоколу.

Так же, как и в случае службы каталога, сферой деятельности сервера приложений является обслуживание подключающихся к нему клиентов (политика пользователей, механизм транзакций и т. п.), а также некоторые специфические функции, связанные с обслуживанием приложений (время жизни приложения, способ хранения информации, которой обмениваются приложения, и т. д.). Как правило, серверы приложений включают в себя и некоторые механизмы защиты от взлома и сбоев оборудования. Среди них средства шифрования, аутентификации, резервного копирования и многое другое. Здесь мы привели список основных компонентов, образующих Web-приложение для небольшого предприятия (несколько сотен рабочих мест). Вопрос о выборе модели данных и языка для ее описания тесно связан со спецификой решаемой задачи. Поэтому для начала рассмотрим задачи, решаемые с помощью XML.

Круг задач, решаемых с помощью XML

Можно выделить три основных области применения языка XML, если как отправной точкой воспользоваться существующими на сегодняшний день типами моделей данных.

Воспользовавшись классификацией баз данных, выделим среди них те, которые наиболее часто используются для решения практических задач:

- одно- или многомерный список (реляционная база данных);
- иерархическая структура (дерево);
- сетевая структура (граф).

Определим первый класс задач, для которых можно воспользоваться услугами XML. Если данные, с которыми предстоит работать Web-приложению, можно описать с помощью одного из обозначенных типов, и по каким-либо причинам нет необходимости привлекать соответствующее программное обеспечение (например, вследствие малого объема данных, или их простой структуры, или дороговизны этого ПО и т. д.), то для полной или частичной организации информации вполне подойдет язык XML.

Вторым классом задач является ситуация, когда XML берет на себя управление какой-либо частью Web-приложения. Даже при создании Web-сайта, который состоит хотя бы из нескольких HTML-документов, иногда бывает необходимо эти документы как-то упорядочить. Примером, когда использование XML может сильно сэкономить время, является создание карты сайта. Другим примером может быть, в частности, построение пользовательского интерфейса, практической реализации которого посвящена эта книга. Таким образом, в этих и им подобных задачах язык XML служит одним из посредников между массивом данных и к ним обращающимися пользователями или приложениями.

Говоря о том, что главной особенностью языка XML является его способность работать с документом как с множеством динамических данных, а не просто как с текстом, мы вовсе не имели в виду, что эти данные не могут представлять собой текст. Хранение информации в виде текста или гипертекста наиболее естественно для человека, и, вместе с тем, такой способ хранения не укладывается ни в одну из схем моделей данных, описанных ранее. Для более четкого представления того, о чем идет речь, рассмотрим такой пример. Представим себе набор текстов на английском языке, каждый из которых иллюстрирует использование в контексте некоторых содержащихся в нем слов. Изучающий английский язык может на основе этих текстов получить представление о корректном использовании английских слов с неоднозначным переводом (идиом).

Если посмотреть на Web-приложение, осуществляющее таким образом обучение английскому языку нескольких учащихся одновременно, то можно выделить следующие его услуги своему администратору:

- добавление текстового документа;
- удаление документа;
- пометка слова, содержащегося в документе, как ключевого (идиомы);
- удаление этой пометки;
- поиск документа по одному или нескольким ключевым словам (идиомам).

Такого рода задача является примером третьего класса задач, решаемых с помощью XML. Очевидно, что таким образом организованные данные плохо поддаются формализации, и если ключевые слова образуют древовидную структуру, то как быть с остальным текстом? Сразу сознаемся, что язык XML рассматривает и текст и ключевые слова в виде отдельных узлов дерева, но при этом формализованные данные имеют легко читаемый текстовый вид. Кроме того, в подобном случае, чтобы как-либо оперировать текстом или его частями, нет необходимости приводить эти данные к другому, специальному виду. Таким образом, обработка текстовых документов является основным классом задач, для которых язык XML изначально проектировался и, следовательно, подходит лучше всего.

Ключевые вопросы:

1 Преимущества XML;
2 Сопоставление XML и HTML;
3 Проект XML: прорисовка формы;
4 Стандартный обобщенный язык разметки;
5 Расширяемый язык встраивания;

6 Расширяемый язык запросов;
7 XML среди других серверных технологий.

Литература:

1 Кузнецов И.Н. Научное исследование : методика проведения и оформления/ И. Н. Кузнецов . -3-е изд., перераб. и доп.. -М.: Дашков и К, 2008.-458 с
2 Рузавин Г.И. Методология научного познания : учеб. пособие : рек. УМЦ/ Г. И. Рузавин. -М.: ЮНИТИ-ДАНА, 2009.-288 с.
3 Кожухар В.М. Основы научных исследований: учеб. пособие/ В.М.Кожухар, -М.:Дашков и К, 2010. -216 с.

2. Методические материалы к выполнению лабораторных работ

Лабораторная работа 1. Основы работы с VisualStudio. Отладка приложений в VisualStudio.

Цель данной лабораторной работы – получение практических навыков отладки параллельных MPI программ в среде Microsoft Visual Studio 2005. При этом будет предполагаться, что на рабочей станций станции установлен Compute Cluster Server SDK, и, следовательно, используется реализация MPI от компании Microsoft (библиотека MS MPI).

Локальный запуск параллельной программы вычисления числа Пи на рабочей станции

• Скомпилируйте проект параллельного вычисления числа Пи (parallelpi) в соответствии с инструкциями лабораторной работы “Выполнение заданий под управлением Microsoft Compute Cluster Server 2003” в конфигурации Release,

• Откройте командный интерпретатор (“Start->Run”, введите команду “cmd” и нажмите клавишу “Ввод”),

• Перейдите в папку, содержащую скомпилированную программу (например, для перехода в папку

“D:\Projects\senin\mpi_test\parallelpi\Release” введите команду смены диска “d:”, а затем перейдите в нужную папку, выполнив команду “cd D:\Projects\senin\mpi_test\parallelpi\Release”),

Для запуска программы в последовательном режиме (1 процесс) достаточно просто ввести имя программы и аргументы командной строки. Например, “parallelpi.exe 1500”,

Для тестового запуска программы в параллельном режиме на локальном компьютере введите “mpiexec.exe –n <число процессов> parallelpi.exe <параметры командной строки>”. В нашем случае параметром командной строки программы вычисления числа Пи является число интервалов разбиения при вычислении определенного интеграла.

Лабораторная работа 2. Основы работы с XCode. Отладка приложений в XCode.

Xcode — это пакет инструментов для разработки приложений под Mac OS X и iPhone OS, разработанный Apple. Xcode IDE предоставляет вам все, что нужно: от профессиональных редакторов, с функцией автозавершения кода и Cocoa рефакторинга, до настройки open-source компиляторов. Xcode IDE разработан с нуля, чтобы вы могли воспользоваться всеми возможностями Cocoa и новейшими технологиями Apple.

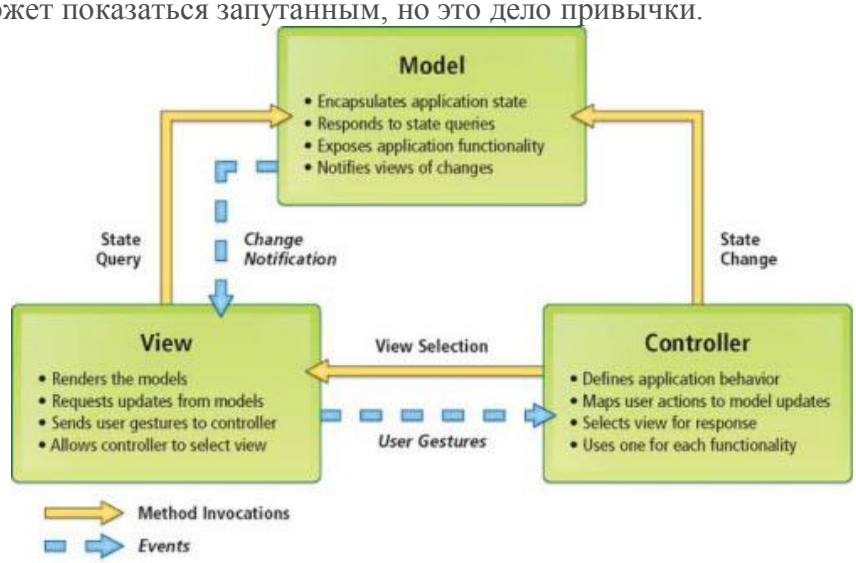
Инструмент разработки для IOS и Mac OS X программирования называется Xcode. Если вы работаете в OS X Lion, вы можете найти Xcode и все соответствующие пакеты бесплатно в *Mac App Store*.

После завершения установки, запустите Xcode, и появится экран приветствия. Отсюда вы можете загрузить старый проект или создать новый. Сейчас вы должны нажать кнопку «*Create a new Xcode project*», и в новом окне вам будет предложено несколько вариантов. Выберите «*Single View Application*» и нажмите «Next». Вы можете назвать новое приложение, например, Test (желательно без пробелов), затем в поле «Company Identifier» введите любое слово, например, MyCompany, выберите папку и нажмите кнопку «Save».

Xcode создаст директорию и откроет новое окно для работы. Вы увидите список файлов, а папка, которая названа в честь вашего приложения, будет первой.

С новым Xcode 4,2 у нас есть два варианта дизайна элементов внешнего интерфейса. Классический *xib/nib* формат является стандартным для Mac OS X и приложений IOS, он требует от вас создавать каждый раз новый вид страницы. Кроме того, вы увидите и файлы *.h и .m.* Это сокращенные имена файлов для заголовков и выполнения. Эти файлы находятся там, где вы пишете все Objective-C функции и переменные, необходимые для запуска вашего приложения. Теперь нужно объяснить, как Xcode работает с *MVC (Model, View, Controller)*, что является причиной того, что нам нужно 2 файла для каждого контроллера.

Чтобы понять, как приложение работает, вам необходимо понять архитектуру его программирования. С Model, View, Controller (MVC) в качестве основы, Xcode может разделить все ваши дисплеи и коды интерфейса, исходя из вашей логики и функций обработки. На первый взгляд MVC может показаться запутанным, но это дело привычки.



Чтобы было легче понять, разберём каждый объект:

- *Модель (Model)* — Вмещает все ваши логические и основные данные. Это: переменные, подключения к внешним RSS-каналам или изображения, подробные функции и числовую информацию. Этот слой полностью отделяется от вашего визуального оформления, так что вы можете легко изменить вид дисплея, и у вас все равно останутся те же данные.
- *Вид (View)* — экран или стиль отображения вашего приложения. Список таблиц, профиль страницы, статьи, аудио-плеер, видео плеер — все эти примеры экранов. Вы можете изменить свой стиль и удалять элементы, но вы будете работать с теми же данными в вашей модели.
- *Контроллер (Controller)* — выступает в качестве посредника между ними. Вы подключаете объекты вида в ViewController, который передает информацию модели. Так что пользователь может нажать на кнопку, и зарегистрироваться в вашей модели. Затем выполнить выход из системы, и через тот же контроллер передать сообщение «вы успешно вышли из системы!»

В основном ваша модель содержит всю информацию и функции, которые вам понадобятся для отображения на экране. Но модели не могут взаимодействовать с экраном, зато могут виды. Виды – это, в основном, все визуальные эффекты, и они могут только извлекать данные через ViewController. Контроллер на самом деле — это сложный способ передачи данных через интерфейс. Таким образом, вы можете обновлять дизайн, при этом не теряя какой-либо функциональности.

Обладая этими знаниями, вы не должны столкнуться с трудностями при попытке создать новое приложение. Как упоминалось ранее, Objective-C является основным языком программирования, который вы будете использовать для разработки приложений. Он построен

на языке C, с обновленным синтаксисом и несколькими дополнительными парадигмами. Понадобится много времени, чтобы познакомиться с языком.

Лабораторная работа 3. Основы работы с DashCode. Отладка приложений в DashCode.

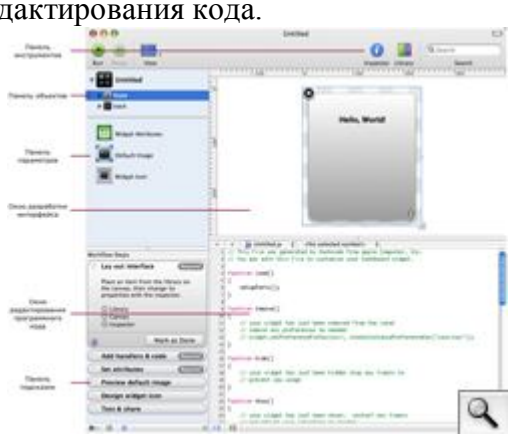
Сразу же после загрузки Dashcode предложит вам на выбор 8 шаблонов, один из которых, как и положено, пустой. По иконкам и названиям разделов не трудно догадаться об их предназначении: в шаблонах предусмотрена работа с RSS, подкастами, фотокастами, счетчиками и т.д.

Стартовое окно Dashcode: выбор шаблона создаваемого виджета

В диалоговом окне кроме кнопок Choose (Выбрать) и Close (Закрыть) доступно еще две кнопки: Open Existing... (Открыть существующий) и Import... (Импортировать). Обе кнопки позволяют открыть уже существующие виджеты. Разница в том, что кнопка Open Existing непосредственно открывает уже существующий виджет и вносить какие-либо изменения вы будете именно в нем. Напротив, при нажатии на кнопку Import создается копия выбранного виджета. Кстати, виджеты, как поставляемые вместе с операционной системой, так и проинсталлированные вами, хранятся по следующему адресу: Library\Widgets.

Выберите пустой шаблон (Custom) и нажмите на кнопку Choose — вашим глазам откроется рабочее окно Dashcode и самый простой на свете виджет, единственным предназначением которого является приветствование окружающей среды.

Основное окно программы можно условно разделить на несколько разделов: Панель инструментов, Панель объектов, Панель параметров виджета, Панель подсказок, Окно разработки интерфейса и Окно редактирования кода.



Рабочее окно среды разработки виджетов Dashcode

Ну чтож, начнем препарирование интерфейса Dashcode по порядку.

На Панели подсказок отображается последовательность шагов, необходимых для создания полноценного виджета:

- разработка интерфейса;
- добавление к объектам интерфейса обработчиков событий и их описание;
- установка параметров виджета;
- регулировка внешнего представления виджета по умолчанию;
- создание иконки виджета;

- тестирование и создание дистрибутива виджета.

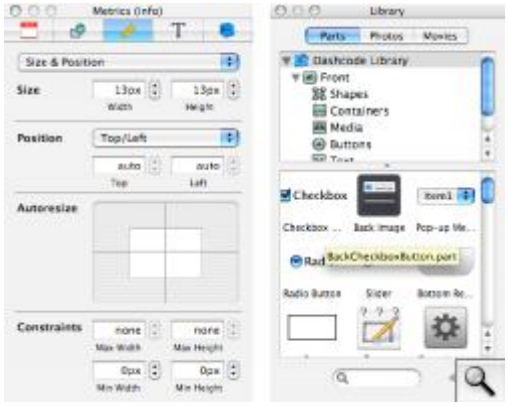
Поскольку теперь вы уже знаете эту последовательность, Панель подсказок вам больше не нужна. Выберите из меню команд пункт View > Steps и панель исчезнет.

С помощью Панели объектов вы можете просматривать и выбирать объекты, отображаемые на лицевой (front) или задней (back) сторонах виджета.

Посредством Панели параметров виджета вы можете настроить: внешнее представление виджета по умолчанию, пиктограмму виджета, а также разнообразные параметры, такие как разрешение сетевого доступа, подключение дополнительных плагинов и управление локализацией.

Как и следует из названия с помощью Окна разработки интерфейса настраивается внешнее представление виджета, а его функциональность определяется посредством окна редактирования программного кода.

Есть еще два вспомогательных окна, не упомянуть которые просто невозможно. Это традиционный для эпповских программ Инспектор (Inspector) и окно Библиотека (Library). С помощью Инспектора настраиваются индивидуальные параметры выбранных объектов и добавляются обработчики событий. Окно Библиотека предоставляет доступ не только к различным объектам, которые можно использовать при разработке интерфейса виджета, но и к фотографиям и видеороликам хранимым в библиотеках iPhoto и iMovie соответственно.



Окна приложения Dashcode Инспектор (Inspector) и Библиотека (Library)

Будем считать, что знакомство с интерфейсом Dashcode прошло успешно.... Кстати, а вы знаете что именно представляет собой виджет? Из чего он состоит? Так, кажется настало время небольшого ликбеза!

Из строки меню выберите команду View > Files. В левом нижнем углу появится список файлов, из которых и состоит виджет. Вы можете видеть, что структуру виджета состовляют кроме графических файлов: html-страница, файл каскадных таблиц и документ с расширение .js.

Внешнее представление виджета зависит от html-страницы, файла каскадных таблиц и графических файлов. А файл с расширением .js хранит программный код виджета, написанный на языке Javascript, и определяет его функциональность.

Отлично, теперь мы предупреждены, и, следовательно, вооружены. Можем начинать разработку игры. Однако преджде следует определится с поставленной задачей. Она не сложная: создать виджет-игру, в которой нужно будет за определенный промежуток времени как можно больше раз попасть мышкой в подвижную мишень.

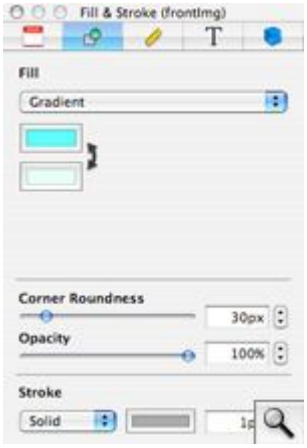
Теперь можно приступать к созданию виджета!

Поле боя

Прежде всего создадим интерфейс игры. Потянув с помощью мыши за правый нижний угол лицевой стороны виджета вы можете изменить его размеры. Однако, есть более эффективный способ задать размеры приложения — с помощью Инспектора. Щелкните мышью на разделе front Панели Объектов. Откройте в Инспекторе вкладку Metrics (Разме-

ры) и введите в поля Width (Ширина) и Height (Высота) значения 800px и 500px соответственно.

Теперь щелкните мышью внутри серого прямоугольника (вашего будущего виджета) или выберете на Панели Объектов в разделе front объект frontImg. Затем откройте вкладку Attributes (Параметры). В поле ID (Идентификатор) отображается текущее название объекта, с помощью которого к нему можно обращаться и получать доступ к его свойствам. Замените название frontImg на poverhnost. Перейдите к вкладке Fill & Stroke (Заливка и Контур). Убедитесь, что к объекту poverhnost применена градиентная заливка. С помощью выпадающего меню Fill вы можете выбрать другие способы заливки фона виджета. Я же просто изменю цвета градиентной заливки.



Вкладка Инспектора Заливка и Контур (Inspector Fill & Stroke)

Также на вкладке Fill & Stroke присутствуют два движка: Corner Roundness (Скругление углов) и Opacity (Непрозрачность). Используя первый вы можете изменять степень округлости углов виджета (в своем примере я определили ее в 30px), а с помощью второго регулировать прозрачность заливки.

Стиль, толщину и цвет контура виджета вы можете определить в разделе Stroke все той же вкладки Fill & Stroke. Я выбрал в выпадающем меню стиль Bevel, а толщину контура установил 7px.

Так, с этим закончили. Теперь нужно добавить новые объекты. В компанию к надписи “Hello, World!” перенесите из Библиотеки еще четыре текстовых блока (объект text). Измените название (поле ID вкладки Attributes Инспектора) надписи созданной по умолчанию на nadpis, а названия добавленных вами объектов на static_nadpis_ochki, static_nadpis_schetchik, nadpis_ochki и nadpis_schetchik.

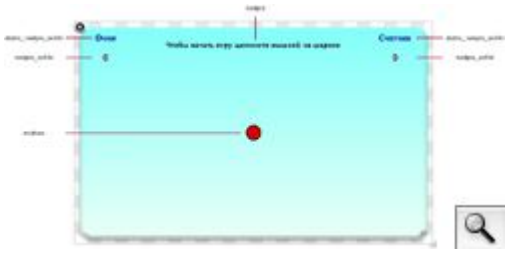
Дважды щелкните мышкой внутри текстового блока “Hello, World!” и введите следующий текст “Чтобы начать игру щелкните мышкой на шарике”. На вкладке Metrics установите ширину текстового блока в 550px. Это необходимо, так как в процессе игры текст надписи изменится, его станет больше и он не сможет целиком помещаться в одной строке.

Теперь откройте вкладку Text (Текст) Инспектора.

Вкладка Инспектора Текст (Inspector Text)

В разделе Character (Символ) вы можете изменить шрифт, стиль начертания, цвет и размер выбранного текстового блока. Также в этом разделе можно добавить тень к тексту. С помощью вкладки Alignment & Spacing (Выравнивание и Интервал) регулируются: стиль выравнивания текста, межсимвольный, межсловесный и межстрочный интервалы.

Установите выравнивание всех текстовых блоков по центру, увеличте размер шрифта четырех добавленных текстовых блоков, измените в них текст и расположите их так, как показано на рисунке. При желании вы можете также как и я изменить цвет текстовых блоков.



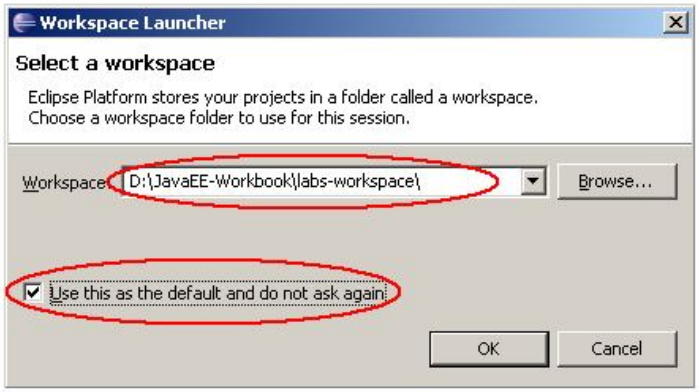
Окно разрабатываемой игры

Наверно вы обратили внимание, что на иллюстрации уже присутствует наша будущая мишень — красный круг. Вам необходимо добавить из Библиотеки в виджет объект Oval Shape (Овал). В своей версии игры я залил его красным цветом и установил толщину контура в 2px. Расположите мишень по центру виджета.

Ну вот и все — мы закончили разработку интерфейса!

Лабораторная работа 4. Основы работы с Eclipse.

При первом запуске Eclipse предлагает выбрать каталог рабочей области (Workspace). Это каталог, в котором будут храниться все файлы, относящиеся к проектам – настройки проектов, исходные файлы программ, результаты компиляции и сборки и т.д.



Укажите каталог рабочей области, выберите флажок Use this as the default… и нажмите ОК.

При дальнейшей работе, переключение между рабочими областями, а также создание новой рабочей области выполняется с помощью команды меню File/Switch Workspace.

Прежде чем приступить к созданию проекта Eclipse, познакомимся поближе с основными элементами Eclipse Workbench.

Eclipse Workbench состоит из:

- перспектив (perspectives);
- представлений (views);
- редакторов (editors).

Перспектива – группа представлений и редакторов в окне Workbench. Количество перспектив в одном окне Workbench не ограничено. Так же и перспектива может содержать неограниченное количество представлений и редакторов. В одном окне каждая из перспектив может иметь различные наборы представлений, но все перспективы делят между собой один и тот же набор редакторов.

Для подключения перспективы используется команда основного меню Window/Open Perspective. Перечень открытых перспектив по умолчанию отображаются в правой верхней части экрана. С помощью этого же списка выполняется переключение между перспективами:

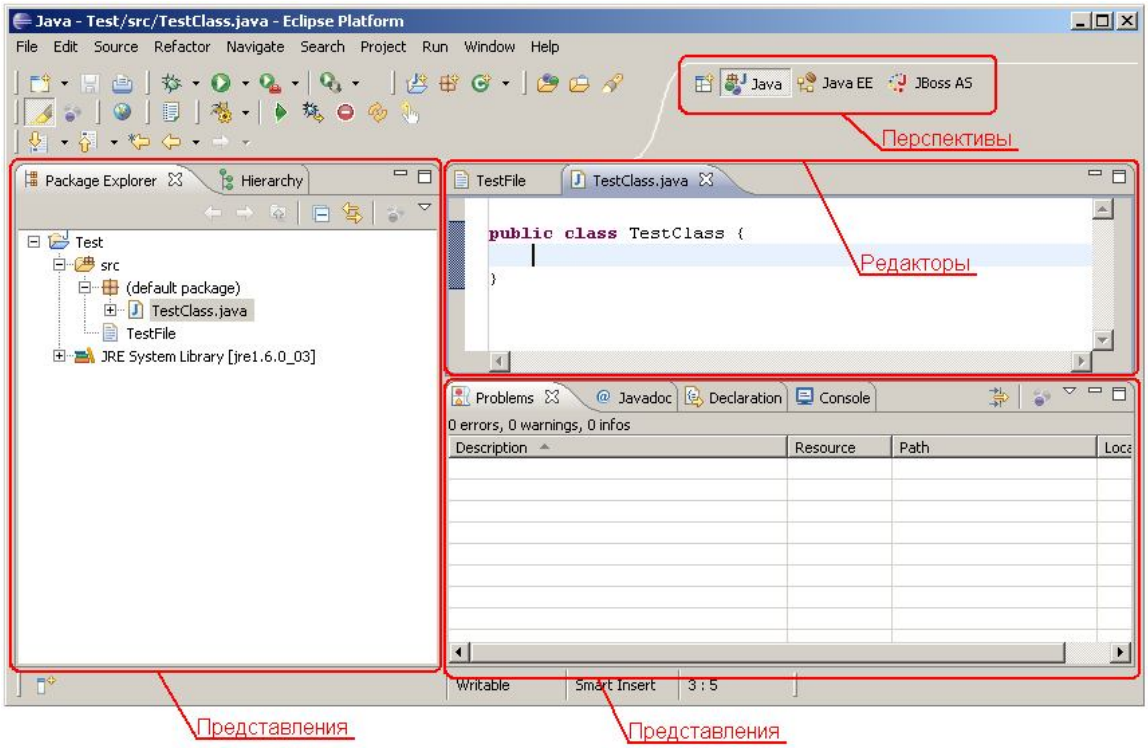
Представление – это визуальный компонент Workbench. Обычно он используется для перемещения по иерархии некоторой информации (например, структуры проекта), открытия редакторов, просмотра результатов выполнения и т.д. Все изменения, сделанные в представлениях, немедленно сохраняются. В большинстве случаев, в окне Workbench может существовать только один экземпляр представления некоторого типа.

Многие представления входят в состав тех или иных перспектив и отображаются при их открытии. Тем не менее, с помощью команды меню Window/Show View можно открыть любое представление, независимо от текущей перспективы.

Восстановить набор представлений текущей перспективы можно с помощью команды Window/Reset Perspective.

Редактор также является визуальным компонентом Workbench. Используется для просмотра и редактирования некоторого ресурса, например Java-класса или файла, содержащего SQL-запросы. Изменения, делающиеся в редакторе, подчиняются модели жизненного цикла «открыть-сохранить-закрыть» (an open-save-close lifecycle model). В Workbench может существовать несколько экземпляров редакторов.

Редакторы того или иного типа обычно отображаются автоматически при двойном щелчке на соответствующий ресурс (например, Java-класс). Сохранение данных выполняется с помощью команды меню File/Save, либо нажатием на клавиши Ctrl-S.



В данный момент времени может быть активным только одно преставление или редактор. Активным является представление или редактор у которого подсвечен заголовок. На активный элемент будут воздействовать общие операции вырезки, копирования и вставки (cut, copy, paste). Также активный элемент обуславливает содержимое строки состояния (status line). Если закладка редактора белая, то это означает, что данный редактор неактивен, однако представления могут отображать информацию, полученную из редактора, бывшего активным последним.

Создание проекта и ресурсов проекта

Проект Eclipse – это совокупность служебных и конфигурационных файлов Eclipse и ресурсов проекта, относящихся к конкретному разрабатываемому приложению или подсистеме приложения. Физически на диске проект представляет собой каталог по имени проекта внутри каталога рабочей области, внутри которого содержатся файлы – составляющие проекта.

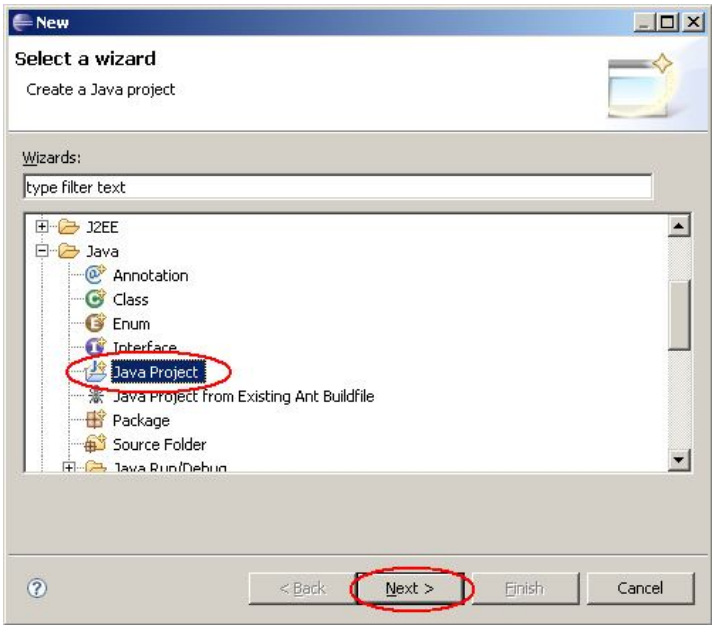
Под *ресурсом проекта* понимаются любые элементы, из которых строится приложение – Java-классы, Java-библиотеки, HTML-страницы, изображения, текстовые файлы, наборы SQL-скриптов и т.д. Строго говоря, сам проект также является ресурсом. Физически каждый ресурс проекта представлен каталогами и файлами соответствующего типа.

Существует несколько способов создания проектов и ресурсов проекта. Основным из них является команда основного меню File/New. С помощью подпункта Other можно отобразить все типы ресурсов, доступных для создания. Для удобства выбора типы создаваемых проектов и ресурсов делятся на условные категории, обычно в зависимости от типа разрабатываемого приложения. Например, в категории Java размещаются обычные Java-проекты, Java-классы, интерфейсы, пакеты и т.д., а в категории Web – Web-проекты, HTML-страницы, JSP-страницы и т.д.

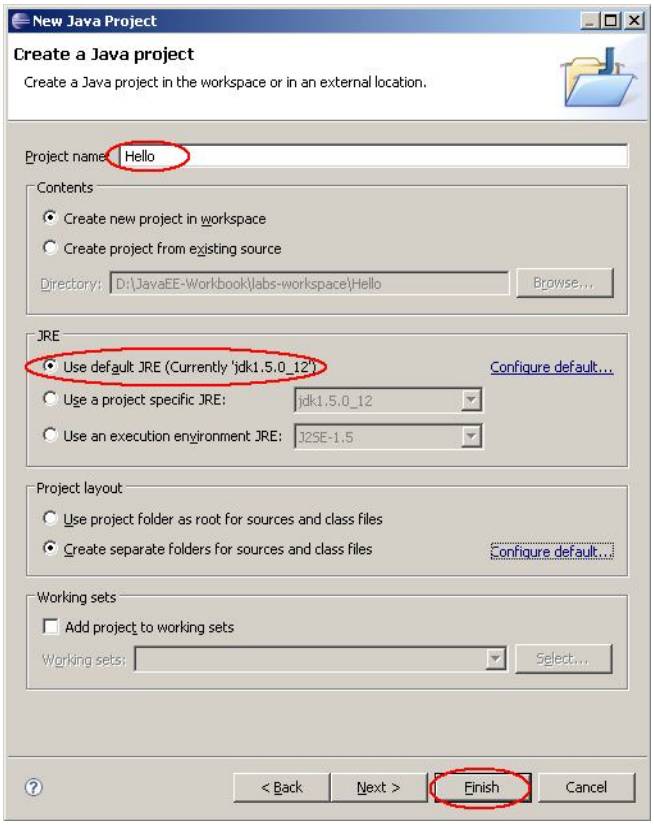
1) Закройте перспективу Welcome, появившуюся на экране после первого запуска Eclipse.

2) Выберите в меню File/New/Other.

3) Выберите категорию Java, а в ней Java Project и нажмите Next.



4) Введите имя проекта – Hello. Затем, если в разделе JRE не указана или указана неверная настройка JRE, выберите Configure default… и добавьте путь к JDK, установленной на предыдущем этапе (например, C:\jdk1.5.0_12).



5) Нажмите Finish.

6) Eclipse предложит открыть перспективы, связанные с разработкой Java-приложения. Выберите флажок Remember my decision и нажмите Yes.

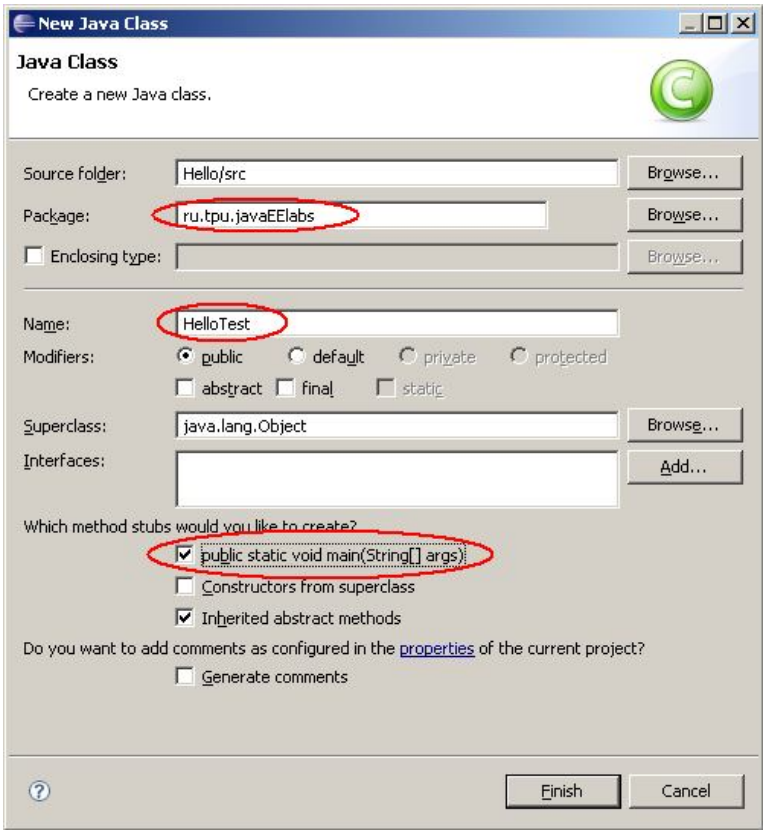
7) В результате создания проекта автоматически открывается перспектива Java, в левой части окна открывается представление Package Explorer, в котором отражается новый проект Hello, содержащий подкаталог src, предназначенный для хранения исходных файлов программы.

8) Далее, нам необходимо добавить новый ресурс проекта – обычный Java-класс:

9) В представлении Package Explorer нажмите правой кнопкой мыши на значок проекта Hello и выберите New/Class. То же самое действие можно выполнить с помощью команды File/New/Other, затем выбрав категорию Java и тип ресурса Class.

10) В появившемся окне укажите имя пакета и имя нового класса. Имя пакета может состоять из нескольких уровней, разделенных точкой. Также выберите флажок public static void main(…) для автогенерации main-метода.

11) Нажмите Finish



12) В методе main запишите команду вывода сообщения на экран:

```
public static void main(String[] args) {
        System.out.println("Hello Java");
    }
```

13) Сохраните файл, нажав на клавиши Ctrl-S.

Лабораторная работа 5. Построение консольных приложений

Будем рассматривать построение консольного приложения при помощи библиотеки GLUT или GL Utility Toolkit, получившей в последнее время широкое распространение. Эта библиотека обеспечивает единый интерфейс для работы с окнами вне зависимости от платформы, поэтому описываемая ниже структура приложения остается неизменной для операционных систем Windows, Linux и многих других.

Функции GLUT могут быть классифицированы на несколько групп по своему назначению:

- Инициализация
- Начало обработки событий
- Управление окнами
- Управление меню
- Регистрация вызываемых (callback) функций
- Управление индексированной палитрой цветов
- Отображение шрифтов
- Отображение дополнительных геометрических фигур (тор, конус и др.)

Инициализация проводится с помощью функции

glutInit (int *argcp, char **argv)

Переменная argcp есть указатель на стандартную переменную argc описы-ваемую в функции main(), а argv – указатель на параметры, передаваемые программе при запуске, который описывается там же. Эта функция проводит необходимые начальные действия для построения окна приложения, и только несколько функций GLUT могут быть вызваны до нее. К ним относятся:

glutInitWindowPosition (int x, int y)
glutInitWindowSize (int width, int height)
glutInitDisplayMode (unsigned int mode)

Первые две функции задают соответственно положение и размер окна, а последняя функция определяет различные режимы отображения информации, которые могут совместно задаваться с использованием операции побитового “или” (“ | “) :

GLUT_RGBA Режим RGBA. Используется по умолчанию, если не
указаны явно режимы GLUT_RGBA или GLUT_INDEX.
GLUT_RGB То же, что и GLUT_RGBA.
GLUT_INDEX Режим индексированных цветов (использование
палитры). Отменяет GLUT_RGBA.
GLUT_SINGLE Окно с одиночным буфером. Используется по
умолчанию.
GLUT_DOUBLE Окно с двойным буфером. Отменяет GLUT_SINGLE.
GLUT_DEPTH Окно с буфером глубины.

Это неполный список параметров для данной функции, однако для большин-ства случаев этого бывает достаточно.

Двойной буфер обычно используют для анимации, сначала рисуя что-нибудь в одном буфере, а затем меняя их местами, что позволяет избежать мерцания. Буфер глубины или z-буфер используется для удаления невидимых линий и поверхностей.

Функции библиотеки GLUT реализуют так называемый событийно-управляе-мый механизм. Это означает, что есть некоторый внутренний цикл, который запускается после соответствующей инициализации и обрабатывает, один за другим, все события, объявленные во время инициализации. К событиям относятся: щелчок мыши, закрытие окна, изменение свойств окна, передвиже-ние курсора, нажатие клавиши, и "пустое" (idle) событие, когда ничего не происходит. Для проведения периодической проверки совершения того или иного события надо зарегистрировать функцию, которая будет его обрабаты-вать. Для этого используются функции вида:

void glutDisplayFunc (void (*func) (void))
void glutReshapeFunc (void (*func) (int width, int height))
void glutMouseFunc (void (*func) (int button, int state, int x, int y))
void glutIdleFunc (void (*func) (void))

То есть параметром для них является имя соответствующей функции задан-ного типа. С помощью glutDisplayFunc() задается функция рисования для окна приложения, которая вызывается при необходимости создания или восстано-вления изображения. Для явного указания, что окно надо обновить, иногда удобно использовать функцию

void glutPostRedisplay (void)

Через glutReshapeFunc() устанавливается функция обработки изменения размеров окна пользователем, которой передаются новые размеры.

glutMouseFunc() определяет обработчика команд от мыши, а glutIdleFunc() задает функцию, которая будет вызываться каждый раз, когда нет событий от пользователя.

Контроль всех событий происходит внутри бесконечного цикла в функции

void glutMainLoop (void)

которая обычно вызывается в конце любой программы, использующей GLUT.

Структура приложения, использующего анимацию, будет следующей:

```
#include <GL/glut.h>
void MyIdle(void){
//--Код, который меняет переменные, определяющие следующий кадр --//
....
};
void MyDisplay(void){
//-- Код OpenGL, который отображает кадр --//
```

```
....
//-- После рисования переставляем буфера --//
glutSwapBuffers();
};

void main(int argcp, char **argv){
….
//-- Инициализация GLUT --//
glutInit(&argcp, argv);
glutInitWindowSize(640, 480);
glutInitWindowPosition(0, 0);
//--Открытие окна--//
glutCreateWindow("My OpenGL Application");
//-- Выбор режима: Двойной буфер и RGBA цвета --//
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
//-- Регистрация вызываемых функций --//
glutDisplayFunc(MyDisplay);
glutIdleFunc(MyIdle);
//-- Запуск механизма обработки событий --//
glutMainLoop();
};
```

В случае, если приложение должно строить статичное изображение, можно заменить GLUT_DOUBLE на GLUT_SINGLE, так как одного буфера в этом случае будет достаточно, и убрать вызов функции glutIdleFunc().

Лабораторная работа 6. Особенности построения приложений для WEB

Для построения клиентского приложения необходимо сгенерировать интерфейсную APL. Генерация выполняется с помощью AX Exprorer CTD – встроенной утилиты. Подробно работа с ней описана в другой статье (Centura Team Developer – ActiveX компоненты: принципы и технология применения), поэтому мы здесь не будем на этом останавливаться. Отмечу только, что генерация основывается на выборе библиотеки типов (*.tlb), которая автоматически генерируется совместно с созданием исполнимого модуля COM сервера и помещается в тот же каталог. После выбора соответствующей библиотеки типов для генерации необходимо выполнить полную генерацию всех классов, событий и перечислений. В создаваемое новое клиентское приложение в раздел библиотек должна быть добавлена эта APL. Ниже на рисунке показано окно простейшего клиентского приложения, созданного для иллюстрации основных положений COM. Эта программа суммирует на сервере COM два целых числа, вводимых в полях формы.

Кроме кнопки суммирования в окне приложения выполняется индикация переменной класса, переменной объекта и числа подключенных пользователей. При нажатии на кнопки приложения вызываются функции COM сервера, результаты работы выводятся на экран в соответствующие поля. На рисунке выше представлена работа одного пользователя. На следующем рисунке представлена работа двух клиентских приложений. В этом случае мы видим, что общие переменные класса (Число пользователей и переменная класса) одинаковы, а переменные объекта различны.

Рассмотрим теперь особенности построения клиентских приложений на основе COM технологии. После подключения APL, для данного COM сервера, в раздел классов добавляются классы, которые соответствуют описаниям объектов сервера. Это следующие классы: функциональный класс – samplCom_IISerge для создания интерфейса (вызова функций) и Com Proxy class – samplCom_Iserge для обработки событий.

При необходимости определения обработчиков событий нужно: создать наследника Com Proxy class и создать доступный объект этого класса. На рисунке показан новый класс

App_Serge, который обеспечивает обработку события evEndSumm. Ниже на рисунке дана иллюстрация результатов генерации интерфейсного класса. Дан весь перечень доступных функций-методов (в левой части окна) и развернуто описания функции суммирования – summa (в правой части окна).

Как видно из примера, перед вызовом метода с помощью универсальной функции INVOKE, передаваемые параметры помещаются в стек (PushNumber), а после выполнения функции – вызываются из стека (PopNumber). Затем стек очищается (FlushArgs). Эта функция (summa) сгенерирована автоматически с помощью AX Explorer, и такое связывание называется статическим. Однако, создав необходимые классы и объекты, пользователь может сам в динамике выполнять операции вызова функций. При этом можно обеспечить и динамическое связывание. Для динамического связывания может быть использована переменная типа VARIANT, которая позволяет вызывать функции независимо от типов параметров.

Лабораторная работа 7. Построение приложений с использованием оконных интерфейсов

Для создания оконных приложений удобнее всего использовать класс *Frame*. В иерархии классов он выглядит следующим образом:

```
java.lang.Object
  |
  +----java.awt.Component
         |
         +----java.awt.Container
                |
                +----java.awt.Window
                       |
                       +----java.awt.Frame
```

По своей природе этот класс похож на классы *Applet* и *Panel.* В объекте класса *Frame* можно размещать элементы управления. Класс, производный от *Frame*, может раскрывать интерфейсы **ActionListener** и **ItemListener**.

Простейшее оконное приложение имеет вид:

```
// Простое оконное приложение
import java.awt.event.*;
import java.awt.*;

class simpleFrame extends Frame
{
    public static void main(String[] args)
    {
    simpleFrame a= new simpleFrame("Оконное приложение");
    }

    simpleFrame(String title)
    {
    setTitle(title);
    setSize(400,200);
    show();
    }
}
```

На первый взгляд структура программы не совсем привычна - в функции *main* создается объект того же класса, к которому принадлежит функция *main*. Зато при создании объекта можно вызвать конструктор с параметрами.

Если вы запустите это приложение, то сразу же обратите внимание, что кнопка закрытия окна не работает. Для обработки событий окна нужно писать специальный код. Он может выглядеть, например, так (код располагается в конструкторе)

```
addWindowListener(
    new WindowAdapter()
    {       public void windowClosing(WindowEvent e)
    {
    dispose();
    System.exit(0);
    }
    });
```

Это пример *определения безымянного класса*. Рассмотрим его подробнее. Мы вызываем метод *addWindowListener* для того, чтобы назначить слушателя оконных событий. В качестве параметра создаем объект класса *WindowAdapter*. Но этот класс является абстрактным! Поэтому мы неявно создаем производный от него класс и переопределяем нужные нам методы - в данном случае обработку события закрытия окна (метод *dispose* уничтожает объект *Frame*). После этого останавливаем виртуальную машину *Java* вызовом метода *System.exit(0)*. При компиляции будет создан класс с именем *simpleFrame$1.class*

Лабораторная работа 8. Использование элементов GUI

Реализовать усеченную версию игры «О, счастливчик» в виде последовательности окон. Файл MS Excel содержит базу вопросов-ответов в формате колонок: A – вопрос, B – правильный ответ, CDE – неправильные ответы, F – начисляемые очки за вопрос.

При решении примера вспомним, как мы поступали ранее при зарисовке графиков, создавая множество figure. Теперь наши возможности расширены за счет команд вызова стандартных типов окон – см. MATLAB-> Functions -- By Category-> Creating Graphical User Interfaces-> Predefined Dialog Boxes. И в данном случае не графика довлеет над расчетной программой, а наоборот.

Сценарий программы таков:

1. Спросить пользователя, использовать ли настройки по умолчанию?
2. Yes – задать путь к новой базе, новый шрифт и вид окна ответов, затем показать шкалу прогресса процесса (waitbar);No – самим задать
3. В цикле показывать вопросное окно, верность/неверность ответа и суммарный балл и предложение продолжить

Соответственно после задания пробной базы в Excel (OLbase.xls) представим макет программы Olucky.m:

```
    %Primary function
function Olucky=Olucky()
%Default param
function [OLbasepath,OLFont,OLStyle]=OluckyDef()
OLbasepath=0,OLFont=1,OLStyle=2,
end
%New param
function [OLbasepath,OLFont,OLStyle]=OluckyNew()
OLbasepath=0,OLFont=-1,OLStyle=-2,
end
%Quest/Answer
function Result=OluckyRes(Number,Summa)
Result=9,
end
[a,b,c]=OluckyDef();a,
```

```
[a,b,c]=OluckyNew();b,
c=OluckyRes(3,4),
Olucky=777;
end
```

Разумеется, пока синтаксически верная программа ничего не делает. Начнем заполнять подфункцию OluckyNew. Соответствующие GUI-вызовы основаны на uigetfile, uisetfont, msgbox:

```
function [OLbasepath,OLFont1,OLStyle]=OluckyNew()
[f,fpath]=uigetfile('*.xls','Где вопросы?','C:\');
OLbasepath=[fpath,f];
button=questdlg('Ответы по вертикали?','Кнопки или списки...','Да','Нет','Да');
if (strcmp(button,'Да')), OLStyle=true; else OLStyle=false; end;
h=msgbox('Сейчас вам будет предложено выбрать шрифт',...
'Или шрифты','help','non-modal');waitfor(h);OLFont1=uisetfont(OLFont);
end
```

Отметив, что переменная OLFont есть структура, пишем

```
OluckyDeffunction [OLbasepath,OLFont,OLStyle]=OluckyDef()
OLbasepath='OLbase.xls';OLStyle=true;
OLFont=struct('FontName','Times New Roman','FontUnits','points',...
'FontSize',12,'FontWeight','normal','FontAngle','italic');
end
```

Чтобы код был исполняемым, произведем замену в запускающей части кода.

```
button=questdlg('Применить параметры по умолчанию?','Вы ленивы?','Да','Нет','Да');
[OLbasepath,OLFont,OLStyle]=OluckyDef();
if(strcmp(button,'Нет')), [OLbasepath,OLFont,OLStyle]=OluckyNew(); end;
xlsread(OLbasepath,-1);
```

Составим новую функцию чтения xls-данных и соответственно заменим последнюю строку на вызов этой вложенной фукнции:

```
%Read xls
function [N,Quest,Ans,Weight]=OLread(path)
[Weight,NQuestAns]=xlsread(OLbasepath,'A:F');
N=size(NQuestAns);N=N(1);
Quest=NQuestAns(2:N,1);Ans=NQuestAns(2:N,2:5);N=N-1;
end….
[N,Quest,Ans,Weight]=OLread(OLbasepath);
```

Теперь обратимся к функции, которая должна выполняться в цикле - OluckyRes. Если пользователь решает прервать игру, то она возвращает -1.

```
%Quest/Answer
function Result=OluckyRes(Number,Summa)
message={{};{};{}};
message{1}=strcat('После ',int2str(Number(1)),'-го вопроса');
message{2}=strcat('у вас в кошельке ',int2str(Summa),' тысяч рублей');
message{3}='Не пора ли забрать деньги?';
button=questdlg(message,'Отвечать иль не отвечать?','Да','Нет','Нет');
%QuestN+AnsN+TrueAnsN - все, что знаем о N-м вопросе
AnsN={Ans{Number(1),Number(2)},Ans{Number(1),Number(3)},...
Ans{Number(1),Number(4)},Ans{Number(1),Number(5)}};
QuestN=Quest{Number(1)};TrueAnsN=Ans{Number(1),1};WN=Weight(Number(1));
if (strcmp(button,'Да')),
Result=-1;
elseif (OLStyle)
[ind,ok]=listdlg('ListString',AnsN,'SelectionMode','single',...
```

```
'PromptString',QuestN,'ListSize',[400 100]);
if (ok==0), ind=1; end;
if (strcmp(AnsN{ind},TrueAnsN))
Result=Summa+WN;
else
Result=Summa;
end
else
message=strvcat('Мудрецы говорят:',AnsN{1},'Соглашаясь, нажмите крестик');
button=questdlg(message,QuestN,AnsN{2},AnsN{3},AnsN{4},AnsN{3});
if (strcmp(button,'')), button=AnsN{1}; end;
if (strcmp(button,TrueAnsN))
Result=Summa+WN;
else
Result=Summa;
end;
end;
end
```

По параметру OLStyle определяем, что применяем – listdlg или questdlg. Перед этим, однако, загружаем из основной базы данных вопрос под номером QuestN и соответствующие ответы и цену ответа. Если ответ оказывается правильным, то сумма приращивается на цену ответа, выводясь в переменную Result.

После того, как мы определились с реализацией опроса по некоторому вопросу из базы данных, нужно написать функцию сортировки ответов (чтобы пользователь не привык к правильности первого варианта).

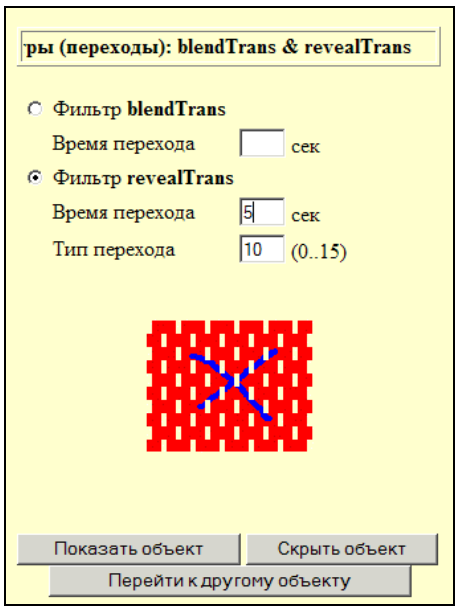
```
%SortAnswer
function Num=OluckySort(Num)
for i=1:20,
ind=floor(1+4*rand(1));ind1=floor(1+4*rand(1));
v=Num(ind1);Num(ind1)=Num(ind);Num(ind)=v;
end
end
```

Теперь в основное тело программы осталось добавить цикл.

```
%Number - номер вопроса по счету, номера ответов в колонке xls... если равен 1, то
%правильный
Summa=0;Numb=[1 2 3 4];
for n=1:N,
Numb=OluckySort(Numb);Number=[n Numb];Result=OluckyRes(Number,Summa);
if (Result<0), break, end;
Summa=Result;
end;
Olucky=Summa.
```

Лабораторная работа 9. Использование визуальных эффектов

Напишите скрипт, демонстрирующий работу динамических фильтров, заголовок оформите в виде бегущей строки.



3. Перечень используемых программных продуктов

Windows XP; MS Office XP; CSS; HTML; Javascript; Ajax; Apache; C++ Builder; XCode; DashCode; Eclipse.

4.Фонд тестовых и контрольных заданий

Современные инструментальные среды – поддержка языков программирования и библиотек. Автоматизация процесса разработки ПО.

Жизненный цикл программного обеспечения. Разработка ПО. Отладка приложения. Подготовка релиза. Инсталляция ПО. Сопровождение ПО.

Этапы работы приложения. Загрузка, создание визуального интерфейса, обработка событий. Файловая структура приложения. Ресурсы приложения. Пользовательский интерфейс. Многоязычная поддержка.

Возможности современных инструментальных систем по созданию приложений. Основные этапы создания приложения. Использование стандартных библиотек и системных ресурсов.

Пример приложения. Структура приложения. Подключение библиотек. Отладка приложения. Установка точек останова. Анализ информации от отладчика. Подготовка приложения к релизу.

Элементы GUI, используемые при построении оконных приложений. Соединение кода с событиями. Цикл обработки событий

Классическая схема построения оконных приложений. Представление, контроллер, модель. Связывание программного кода и событий. Использование имеющихся классов. Наследование.

Представления, элементы для ввода-вывода информации, окна, контроллеры, элементы для вывода графики, видео и изображений. Другие элементы.

Планирование отладки. Использование точек останова. Анализ состояния программы. Просмотр значений переменных и объектов.

Тестирование. Подготовка ресурсов. Создание многоязычных приложений.

Использование стандартного программного обеспечения для создания инсталлятора ПО.

Технические характеристики браузеров. Поддержка браузерами современных стандартов. Различия браузеров.

Стандартные элементы графического интерфейса пользователя. Моделирование интерфейсов локальных приложений.

Использование графики для создания «пользовательских элементов» GUI.

Специальные библиотеки для создания GUI. Создание многовидовых приложений.

CSS переходы и анимация. Управление анимацией. Специальные виды эффектов.

Приложения, использующие данные. Разделение кода и данных. Данные, хранимые локально.

Обзор современных решений.

Построение модели данных. Связь источника данных с потребителем. Источники (базы данных, XML, JSON, key-value Store). События, изменяющие данные.

Перспективы развития Инструментальных средств.

Учебно-методическое обеспечение самостоятельной работы

Карточки с заданиями и методическими указаниями по выполнению практических работ

Учебно-методическое обеспечение самостоятельной работы

Карточки с заданиями и методическими указаниями по выполнению практических работ

СТО СМК 4.2.3.05-2011. Стандарт ФГБОУОВПО «АмГУ». Оформление выпускных квалификационных и курсовых работ (проектов), 2011. – 95 с.