Учебное пособие по дисциплине

ПРОЕКТИРОВАНИЕ КЛИЕНТ-СЕРВЕРНЫХ СИСТЕМ

Практикум

УДК 004.6

ББК 32.973.2-018

Л 68

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Учебно-методическое пособие освещает вопросы основ разработки клиент-серверных информационных систем. Целью пособия является обеспечение студентов по направлению подготовки 09.03.04 − «Программная инженерия» необходимым теоретическим материалом и примерами для освоения практической части дисциплины «Проектирование клиент-серверных систем», а конкретно для выполнения заданий практических занятий.

На современных предприятиях, в организациях, на фирмах и у отдельных предпринимателей, в общем везде, где сотрудники используют более одного персонального компьютера или ноутбука (планшета, смартфона) для обеспечения и выполнения бизнес-процессов, повсеместно присутствуют корпоративные локальные вычислительные сети (ЛВС). Они могут быть организованы как проводные, так и беспроводными. Это делает вычислительную систему организации гибкой и многофункциональной. Возникает возможность организовать, так называемую, распределенную модель обработки информации, когда данныедля совместного доступа могут храниться в общей базе данных (БД), расположенной на отдельном специальном компьютере, называемом сервером. Схема совместного использования вычислительных ресурсов сменила традиционный локальный подход к организации вычислительных систем, когда пользовательские приложения размещались отдельно на каждом рабочем месте и использовались только одним работником. Эпоха однопользовательских систем и т.п. закончилась, ее сменили многопользовательские ресурсы, в том числе и web, т.е. расположенные в сети Internet [1-3].

Очевидно, что распределенная структура вычислительных систем предполагает наличие одного (нескольких ранжированных серверов) сервера – компьютера (нескольких компьютеров) предназначенного для размещения общих ресурсов ЛВС и для организации их использования и множества пользовательских компьютеров или просто пользователей, работающих с ресурсами. Т.о., возникает структура «клиент-серверной» системы.

Распределенные информационные системы и клиент- серверные приложения для обеспечения их функционирования являются признаком современного развития информационных систем. Технологии их разработки довольно разнообразны [4]. Нет принципиальных различий в технологиях разработки внутрикорпоративных сетевых приложений и web – разработками. Другими словами, все, что используется для создания Интернет- приложений распределенной структуры, пригодно и для локального использования внутри сети отдельной организации. Первой широко применяемой технологией разработки такого вида систем является CGI (Common Gateway Interface), которая особенно применяется для создания динамических веб-страниц и служит для обеспечения связи между клиентом и веб-сервером. По этой технологии HTTP запрос, содержащий ссылку на динамическую страницу, поступая на веб- сервер, генерирует новый процесс и запускает нужную прикладную программу. Под эту технологию можно использовать любой язык программирования, работающий с вводом/выводом (C#, Python, Perl, и т.д.). При всех достоинствах основным недостатком этого подхода является производительность, которая ограничивается тем, что для каждого запроса на сервер запускается отдельный процесс, прекращающийся по окончанию выполнения всей программы.

Другой технологией разработки клиент-серверных систем достаточно распространенной и на мобильных платформах выступает Java Servlets или просто – технология сервлетов. Сервлеты – это пример Java- интерфейса, т.е. модуля, находящегося между сервером и клиентом и работающего по принципу «запрос-ответ». Это позволяет оптимизировать работу сервера и организовать выполнение всех запросов в одном процессе, распределив их по потокам.

Еще одной мощной технологией разработки такого вида систем является технология сокетов от Microsoft. Сокеты − это концепция сетевого программирования, когда существуют два вида «программных разъемов» − сокетов: клиентские и серверные.

Серверный сокет как розетка, которая «висит на стене» готовая к работе, в ожидании, когда к ней подключат штекер. Точно так же серверный сокет пе-ре-ходит в режим ожидания подключения на определенном адресе и определен-

ном порту.

Клиентский сокет как вилка, которую втыкают в розетку. Как только «клиент» подключается к серверному сокету, информация начинает передаваться между ними.

Основным достоинством такой технологии является ее простота реализации, а к недостаткам можно отнести ограниченность количества потоков вычислений при организации много потокового взаимодействия.

Еще одной технологией от компании Microsoft является .NET технология. Платформа .NET значительно упростила процесс разработки систем и повысила надежность кода. Стали доступными функции автоматического управления временем жизни объектов, обработка исключений и их отладка, в наличии появились библиотеки, нейтральные к языкам программирования. Набор стандартных базовых классов обеспечивают разработчику доступ к сервисам платформы при использовании любых языков программирования, совместимых с .NET. Common Language Runtime совместно с базовыми классами составляют основу платформы .NET и предлагает разработчикам высокоуровневые сервисы, такие как ADO.NET (усовершенствованный ADO, используемой SOAP и XML с целью обмена данными), ASP.NET (новое поколение ASP, дающий возможность использовать любой язык программирования, совместимый с .NET) и Windows Forms и Web Forms (классы, реализующие локальные и web-ориентированные приложения). Компилирование исходного кода происходит по следующей схеме: создается код на промежуточном языке (Microsoft Intemediate Language).

В первом разделе пособия описаны общие требования к составлению архитектуры любого приложения на примере клиент-серверных систем. Здесь указано назначение таких разработок и общие требования к технологии их создания.

Во втором разделе пособия приводятся разработки клиентской части приложения и серверных частей с помощью языков Java и C#.

Клиентские части систем, как правило снабжаются графическим интерфейсом пользователя. Это удобнее всего выполнить в Windows Form C#. Этому посвящено практическое занятие №1.

Серверы, как правило, не требуют каких-то вмешательств пользователей извне, поэтому логично составить их программы в рамках консольного проекта, т.е. без интерфейса пользователя. Технологией разработок серверных частей является технология сокетов. На ее основе, используя Java и C# создаются простые двухуровневые клиент- серверные приложения с однопоточным сервером (практические занятия №2 и №3 соответственно).

1 Общие положения проектирования клиент-серверных систем

1.1 Структура и назначение клиент-серверных структур

Любой компьютер, сервер, подключённый к локальной или глобальной сети, называется хостом (host − хозяин, принимающий гостей). Клиент/серверная модель представляет собой архитектуру разработки систем, предназначенную для отделения уровня представления данных от их обработки и хранения.

Клиентские хосты запрашивают обслуживание, и хост сервера обеспечивает обслуживание этих запросов. Запрос передается от клиента на сервер через сеть, обработка выполняется на сервере и скрыта от клиента. При этом один сервер может обслуживать несколько клиентов.

На рисунке 2.1 представлена клиент-серверная архитектура, обслуживающая несколько клиентов.

Рисунок 2.1- Несколько клиентов, получающих доступ к серверу

Сервер и клиент не обязательно являются отдельными компонентами оборудования, они могут быть реализованы программами, работающими на одной или различных машинах.

Серверная часть клиент-серверного приложения управляет ресурсами, распределенными среди нескольких пользователей, которые получают доступ к серверу вместе с другими клиентами.

1.2 Основные характеристики клиент-серверных систем

1.2.1 Протоколы

При общении между людьми соблюдаются некоторые правила или протоколы. Например, люди не разговаривают одновременно и непрерывно - никто не смог бы понять, что говорит другой человек, если бы они не следовали определенным правилам. Когда один говорит, собеседник слушает и наоборот. При этом, люди говорят на языке и в темпе, которые понятны их собеседникам.

Когда «общаются» компьютеры, им также необходимо следовать определенным правилам. Данные посылаются от одной машины на другую в форме пакетов (packets). Правила- протоколы определяют процедуру упаковки данных в пакеты, скорость передачи и разборки данных в их исходную форму. Сетевой протокол является набором правил и соглашений, поддерживаемых системами, которые взаимодействуют в сети. Сетевое программное обеспечение обычно реализуется несколькими уровнями протоколов, располагающихся слоями, один над другим.

1.2.2 IP адрес и порт

Сервер любого клиент-серверного приложения, работающего в Интернет, может рассматриваться как набор сокетов, которые обеспечивают дополнительные возможности – обычно называемые сервисами. Примерами сервисов являются электронная почта (email), сетевой удаленный доступ (Telnet) и протокол передачи файлов (FTP -File Transfer Protocol). Каждый сервис связан с

портом (port), который является числовым адресом, через который обрабатывается запрошенный запрос, например, запрос Веб-страницы.

Протокол TCP запрашивает два элемента данных: IP адрес и номер порта. Каким образом происходит, что при вводе адреса URL -Uniform Resource Locator - http://www.skf-mtusi.ru, браузер получает домашнюю страницу указанного ресурса? URL - это стандартизированный способ записи адреса ресурса в сети Интернет. Интернет протокол (Internet Protocol - IP) обеспечивает логический адрес, называемый IP-адресом сетевого устройства. Каждому доменному имени компьютера (www.skf-mtusi.ru) в системе доменных имен (DNS - Domain Name System) соответствует IP-адрес. IP-адреса, используемые в Интернет, имеют определенный формат. Каждый адрес представляется 32-разрядным числом, состоящим из четырех байт (8-разрядных чисел), каждое в диапазоне значений от 0 до 255. СКФ МТУСИ имеет свое зарегистрированное имя, позволяющее www.skf-mtusi.ru представляться определенным IP адресом.

Если номер порта не указан, то используется номер порта по умолчанию для сервиса, примеры которых приведены в таблице 2.1.

Таблица 2.1- Описание портов и систем

Номер порта	Приложение
21	FTP - передает файлы
23	Telnet -обеспечивает удаленный доступ
25	SMTP - доставляет почтовые сообщения
67	BOOTP - обеспечивает конфигурацию во время загрузки
80	HTTP - передает Веб-страницы
109	POP - получает доступ к почтовому сервису на удаленной хосте

Еще раз рассмотрим структуру URL http://www.skf-mtusi.ru.

Первый компонент URL (который является, http) обозначает, что используется протокол передачи гипертекстовых файлов (Hypertext Transmission Protocol - HTTP) для управления HTML-документами. Если файл не задан, большинство wed серверов конфигурируются таким образом, что представляется файл с именем index.html. Таким образом, IP адрес и порт являются определенными либо явной спецификацией всех частей URL, либо применением спецификации по умолчанию.

1.2.3 Сокеты

В клиент-серверной архитектуре систем хост сервера обеспечивает сервисы, аналогичные обработке запросов базы данных. При этом, взаимодействие, которое происходит между клиентом и сервером должно быть надежным, и данные не должны быть потеряны, а также должны быть доступны клиенту в той же последовательности, в которой сервер их отправлял.

Как уже отмечалось, TCP обеспечивает надежный канал соединения точка-точка (point-to-point) для клиент-серверных систем, с помощью которо- го программы клиента и сервера устанавливают соединение и связывают соке- ты. Сокеты - это название программного интерфейса хоста для обеспечения информационного обмена между процессами.

Взаимодействующие процессы могут выполняться как на одном компьютере, так и на различных хостах, соединенных между собой через сеть. Сокеты используются для управления каналом связи между приложениями, установленным через сеть. Каждое TCP-соединение может быть однозначно идентифицировано своими двумя конечными точками. Таким образом, могут быть обеспечены множественные соединения клиента и сервера. После создания сокета, через него выполняется дальнейшее взаимодействие между клиентом и сервером.

1.2.4 Основные отличия клиент-серверных систем

Основными отличиями клиент-серверной модели распределенных вычислений являются:

- в приложениях клиент-сервер большое внимание уделяется созданию на клиентской машине дружественного пользователю интерфейса. Таким образом, пользователь получает полный контроль над расписанием и режимом работы компьютера;
- хотя приложения являются распределенными, в архитектуре клиент-сервер, как правило, используются централизованные корпоративные базы данных. Это позволяет осуществлять совместный доступ к данным;
- как корпоративные пользователи, так и производители отдают предпочтение открытым и модульным системам. Это означает, что пользователю предоставляется более широкий выбор продуктов и большая свобода в объединении оборудования от различных производителей;
- компьютерная сеть является ключевым звеном данной архитектуры. Поэтому вопросы сетевого администрирования и сетевой безопасности при работе с информационными системами данного типа имеют приоритет.

2 Практикум по дисциплине «Проектирование клиент-серверных систем»

2.1 ПЗ-1. Проектирование архитектуры и дизайна клиент-серверных систем

2.1.1 Цели занятия

Практически освоить основные правила проектирования архитектуры и дизайна систем. Практически потренироваться в составлении интерфей- сов клиентской части.

2.1.2 Теоретические основы и пример выполнения

В состав клиент-серверных систем, как составного программного обеспечения (ПО), очевидно, входят серверная и клиентская части. Как правило, обе эти составляющие приложения разрабатываются как интерфейсные продукты, т.е. содержащие специальные модули (блоки) для обеспечения связи человека и вычислительной среды. От этого зависит, в конечном итоге, насколько простым или сложным будет использование приложения, а также его оптимальное функционирование.

Поэтому, важно научиться составлять удобные в использовании и функционально оптимальные клиент-серверные продукты.

Тема данного практического занятия больше относится к реализации клиентских частей приложения. Однако, правила разработки систем относят- ся в равной степени и к созданию серверной части приложения.

Сейчас при разработке ПО большое значение придается его архитектуре. И это не случайно. У любого программного продукта (ПП) есть свое конкретное назначение, которое определяется тем, как с помощью ПП пользователь может удовлетворить потребности в трех областях: пользовательской (интерфейс), предметной (бизнес, исследования, потребительская и т.д.) и в системной с точки зрения структурности продукта. Поэтому, удачная архитектура

приложения делает его гибким, простым в использовании и сопровождении и высоко функциональным продуктом и наоборот.

Под **архитектурой** клиент-серверной системы будем понимать ее структуру, включающую программные элементы, видимые извне свойства этих элементов и взаимоотношения между ними [5].

Типовая архитектура для каждой из частей клиент-серверной системы представлена на рисунке 2.1.

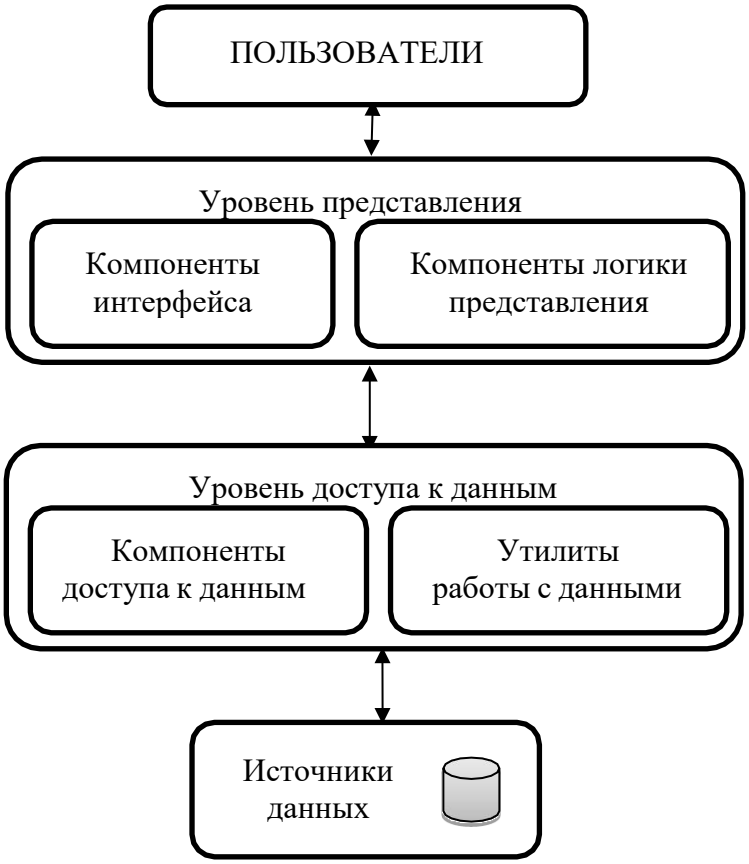

Рисунок 2.1-Типовая структура клиент-серверной системы

Для серверной части характерно еще и то, что ее пользователями могут быть не только люди, а и клиентские части системы.

Дадим краткие пояснения приведенной структуры.

Источники данных – это исходный уровень структуры, т.е. непосредственно данные, которые необходимо передавать или преобразовывать, если

это структура серверной части системы, или можно просто использовать, как говорят: «на стороне клиента». Традиционно, все данные в клиент-серверных системах располагаются на серверной части в виде базы данных, клиентские части системы имеют к ним доступ при обращении на сервер. Однако, структура системы позволяет размещать и хранить данные и на клиентских частях, если это необходимо [6].

Уровень доступа к данным обеспечивает работу с данными, поэтому он включает компоненты доступа к данным и утилиты работы с данными. Под «доступом к данным» подразумеваются алгоритмы их записи, извлечения и обработки. Соответственно этому и выбираются утилиты, их реализующие. Так, если используется база данных, то этот уровень реализуется в виде приложения, управляющего базой данных, если же данные просто размещаются по какому-то принципу на дисках сервера, то и система реализует поисковые алгоритмы доступа к ним.

Уровень представления служит для необходимой обработки данных и представления их в требуемом виде согласно клиентским запросам, а также для обеспечения возможности пользователю вообще работать с системой. Поэтому этот уровень реализует алгоритмы обработки данных и компоненты пользовательского интерфейса. Обычно, это больше касается пользовательских частей системы, т.к. именно через них пользователи работают с клиент-серверными системами, однако и на серверной части возможны минимальные реализации этого уровня для настройки и эксплуатации серверной ее части. Сюда, как показано на рисунке 2.1, входят компоненты интерфейса пользователя и логики представления. Компоненты логики представления необходимы для реализации сценария подачи необходимой информации пользователю. Интерфейсные компоненты, в свою очередь, обеспечивают представление нужной информации пользователю.

Настоящее практическое занятие включает отработку всех описанных уровней реализации системы. При этом, особое внимание уделяется уровню представления, т.к. уровень доступа к данным и сами данные

могут на клиентской стороне клиент-серверной системы отсутствовать вообще.

Клиентскую часть системы необходимо будет разработать, используя Visual Studio- интегрированную среду разработки программных продуктов, снабженную всеми необходимыми средствами написания, отладки и компиляции программ [7]. Воспользуемся языком программирования C# потому, что именно он представляет массу инструментов для создания интерфейсных приложений [8].

2.1.2.1 Основные элементы графического интерфейса пользователя

Для создания клиентских частей клиент-серверных систем исполь- зуется обширная библиотека Form языка C# для систем типа Windows Forms. Принцип составления оконного интерфейса прост- создается пустая форма, которая наследует (использует) библиотеку готовых решений Form. Да- лее, перетаскивая на эту форму необходимые элементы управления и отобра- жения составляется клиентская часть для решения задачи, указанной на практическом занятии.

Элементов управления в указанной библиотеке множество, рассмотрим только часто употребляемые и необходимые для решения поставленных на занятии задач.

Основными элементами форм являются:

– label – метка для отображения справочной информации;
– textBox – текстовое поле для вводимых и выводимых данных;
– button – кнопка для исполнения требуемых действий.

Приведем пример решения одной из задач и покажем как программируются указанные элементы интерфейса.

2.1.2.2 Пример выполнения задания

Задание: даны действительные числа x,y. Вычислить значение функции z=ln(x)-x/y.

Из задания следует, что необходимо будет на форму ввести две переменные и вычислить значение функции для их определенных величин. Это задается видом искомой функции. В данном случае – это логарифм, а областью определения этой функции являются только положительные числа, большие нуля. Т.е., функция имеет смысл только для x,y>0. На форме также, очевидно, должна быть кнопка для выполнения расчетов и метка, либо текстовое поле для отображения результата.

Изначально создается проект Windows Forms с названием Form1 (это имя будет сформировано по умолчанию и может быть изменено). Далее, используя панель инструментов Visual Studio в режиме конструктора размещаем элементы интерфейса на форму (рисунок 2.2).

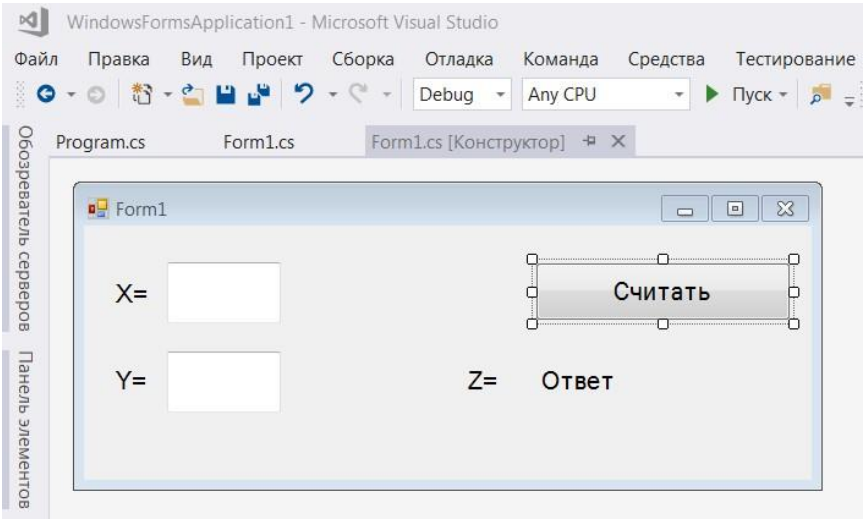

Рисунок 2.2- Конструирование формы интерфейса

При этом необходимый код программы создается компоновщиком форм автоматически.

Далее необходимо создать метод для вычисления функции Z. Дело в том, что весь проект содержит два класса: Program и Form1 (рисунок 2.3).

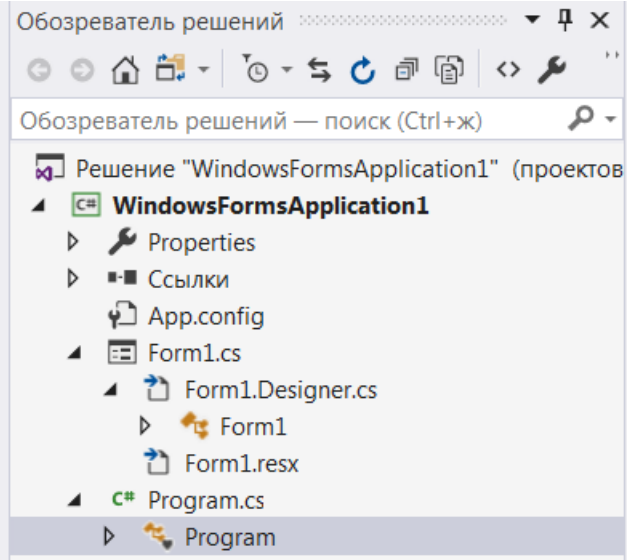

Рисунок 2.3- Состав клиентской части

Program нужен для запуска всего проекта и формирования разработанной формы, он содержит основной метод всего проекта – метод Main(). Класс Form1 содержит все необходимое для задания и отображения элементов формы. Коды этих классов, которые содержатся в файлах Program.cs и Form1.cs, представлены в листинге 2.1.

Листинг 2.1- Коды классов проекта

```
namespace WindowsFormsApplication1
{
    static class Program  // код класса Program
    {
        /// <summary>
        /// Главная точка входа для системы.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
        public static double Metod(int x, int y) // Метод для вы-
числения Z
        {
            double z=0;
            if (y != 0)
```

```
            {
                z = Math.Log(x) - x / y;
            }
            else {
                z = 0;
            }
            return z;
        }
    }
    }


using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form // Код класса Form1
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e) //
Обработчик кнопки
        {
label4.Text = Con-
vert.ToString(Program.Metod(Convert.ToInt16(textBox1.Text),
                            Convert.ToInt16(textBox2.Text)));
        }

    }
    }
```

В классе Program составлен метод Metod(int x, int y), в котором вычисляется искомая функция. При этом, вычисления осуществляются по нажатию на кнопку «Считать», обработчик которой содержится в классе Form1. Обработчик события – это специальный метод (или еще говорят: «функция»), который создается автоматически, если дважды кликнуть по кнопке (или по другому эле-

менту интерфейса) на форме в режиме конструктора. Этот метод как бы «слушает» изменение состояния выбранного элемента интерфейса и если, например, кнопка будет нажата, то выполнится программный код, содержащийся в ее обработчике событий (листинг 2.1).

В обработчике кнопки находится строка присваивания значения метке label4, которая служит для отображения результатов вычислений. Оператор label4.Text указывает на то, что объект label4 обращается к методу Text, чтобы поместить текстовую переменную в метку. Вообще, в C# ввод/вывод возможен только для текста. Поэтому, правая часть оператора присваивания содержит вызов метода Metod(int x, int y), который расположен в классе Program и содержит два параметра- переменные X и У. Эти значения конвертируются сначала из текста текстовых полей textBox1 и textBox2, куда они вводятся на форму, а потом результат выполнения всего метода конвертируется в текст и выводится в метку на форме как результат всей задачи.

2.1.3 Порядок выполнения задач ПЗ

Алгоритм выполнения задач практического занятия следующий:

- выбрать 5 задач по следующему правилу: номер по журналу- первая задача; номер каждой последующей задачи определяется прибавлением цифры 3 к номеру первой задачи, который только что вычислили (если достигнуто окончание списка вариантов задач, то перейти в его начало);
- составить методы (функции) решения всех задач, поместить их в класс Form1 и снабдить пользовательским интерфейсом в Windows Forms. Это клиентская часть будущего клиент-серверной системы, которая разрабатывается на следующих занятиях;
- оформить отчет для всей системы в целом, включив в него задание, блок-схему алгоритма (в электронном виде), текст программы и skrin-shert результата выполнения каждой задачи и представить его на проверку.

2.1.4 Варианты заданий

Решить задачи, используя оператор условного перехода if:

1. Даны действительные числа A,B,C,D. Выяснить, можно ли уместить прямоугольник со сторонами A,B внутри прямоугольника со сторонами C,D.
2. Даны действительные числа x, y, z. Найти минимальное из них.
3. Даны действительные положительные числа A,B,C. Выяснить, пройдет ли кирпич с ребрами A,B,C в прямоугольное отверстие со сторонами x,y.
4. 4.Определить, лежит ли точка D(c,b), где $c=\sqrt{a1+a2}$, $b=a1+0,7a3$, внутри прямоугольника со сторонами 5 и 10 (a1,a2,a3 - произвольные числа)?
5. Выяснить, существует ли треугольник с координатами вершин A(x1,y1), B(x2,y2), C(x3,y3), если да, то найти его площадь.
6. Даны действительные числа A,B,C. Проверить выполняются ли неравенства A<B<C, если да, то присвоить A=B+C иначе A=C-B.
7. Даны действительные числа x,y. Вычислить значение функции z=log(x-y)-x/y
8. На плоскости расположена окружность радиуса R с центром в начале координат. Определить положение точки x с координатами (A,B) относительно окружности.
9. Даны круг радиуса R и квадрат со стороной A. Определить их взаимное положение.
10. Вывести на печать переменные A,B,C в порядке их возрастания
11. Проверить, какие из чисел A,B,C,D принадлежат интервалу (1,25).
12. Даны действительные числа A,B. Если они оба отрицательные, то заменить каждое из них его квадратом, иначе положительные из них увеличить в два раза.
13. Выяснить, существует ли треугольник с координатами вершин A(x1,y1), B(x2,y2),C(x3,y3).
14. Даны действительные числа x,y. Вычислить значение функции z=log(x/y)-1/x

15. Даны действительные числа A,B. Если они оба неотрицательные, то заменить каждое из них его кубом, иначе отрицательные из них заменить их модулями.
16. Даны площадь квадрата S1 и круга S2. Определить поместится ли круг в квадрат и наоборот.
17. На плоскости расположена окружность радиуса R с центром в начале координат. Определить, лежат ли точки A(x1,y1) и B(x2,y2) на окружности.
18. Составить программу вычисления корней системы уравнений с двумя неизвестными методом Крамера.

    $\begin{cases} a1x + b1y = c1 \\ a2x + b2y = c2 \end{cases}$ Проверить, что главный определитель системы не должен быть равен 0.
19. Даны действительные числа A,B,C,D. Выяснить, можно ли уместить прямоугольник со сторонами A,B внутри прямоугольника со сторонами C,D.
20. Вывести на печать переменные A,B,C в порядке их убывания
21. Даны действительные числа x, y, z. Найти максимальное из них.
22. Проверить, какие из чисел A,B,C,D не принадлежат интервалу (3,15).
23. Даны действительные числа x,y. Вычислить значение функции z=ln(x)-x/y.
24. Даны действительные числа A,B. Если они имеют разные знаки, то напечатать их произведение, иначе напечатать их квадраты.
25. Выяснить, существует ли треугольник с длинами сторон A,B,C. Если да, то найти его площадь.
26. Даны действительные числа x,y. Вычислить значение функции z=arcsin(x)-y.
27. Даны действительные числа x,y,z. Получить максимальное из них по модулю.
28. Даны действительные числа x,y. Вычислить значение функции z=arcsin(x+y).
29. На каком из интервалов $(-\infty;k1),(k1;k2),(k2;+\infty)$ лежит точка x? Где k1, k2, x – вводимые произвольные числа, причем k1<k2.

30. Лежат ли обе точки D(a1;b1) и C(a2;b2) внутри круга радиуса R с центром в начале координат? Если такой точки нет, выдать соответствующее сообщение.

2.2 ПЗ-2. Простые двухуровневые клиент-серверные системы в Java

2.2.1 Цели занятия

Выработать умения и навыки по составлению программ простых одноуровневых клиент-серверных систем на основе технологии сокетов.

2.2.2 Теоретические основы и пример выполнения

2.2.2.1 Создание сетевых систем с использованием TCP

Java поддерживает классы и методы, которые позволяют устанавливать соединение с удаленным компьютером, используя протокол TCP. В отличие от UDP, TCP является протоколом, ориентированным на установление соединения, которое гарантирует надежную связь между приложениями клиента и сервера. Взаимодействие с использованием протокола TCP, начинается после установления соединения между сокетами клиента и сервера. Сокет сервера "слушает" запросы на установление соединения, отправленные сокетами кли- ентов, и устанавливает соединение. После установления соединения между приложениями клиента и сервера, они могут взаимодействовать друг с другом.

Java упрощает сетевое программирование, путем инкапсуляции функциональности соединения сокета TCP в классы сокета, в которых класс Socket предназначен для создания сокета клиента, а класс ServerSocket для создания сокета сервера.

2.2.2.2 Идентификация методов классов Socket и ServerSocket

Socket является базовым классом, поддерживающим протокол TCP. Класс Socket обеспечивает методы для потокового ввода/вывода, облегчает выполне-

ние операций чтения и записи в сокет и является обязательным для программ, выполняющих сетевое взаимодействие.

Для создания объектов класса Socket используются следующие конструкторы, определенные в классе Socket:

– public Socket (InetAddress IP_address, int port) - создает объект Socket, который соединяется хостом, заданным в параметрами IP_address и port.
– public Socket (String hostname, int port) – создает объект Socket, который соединяется с хостом, заданным параметрами имя хоста или IP адрес и port, который сервер "слушает".

Некоторые полезные методы класса Socket представлены в таблице 2.1.

Таблица 2.1- Основные методы класса Socket

Метод	Описание
public InetAddress() public getInetAddress()	Возвращает объект InetAddress, который содержит IP-адрес, с которым соединяется объект Socket
public InputStream() public getInputStream()	Возвращает входной поток для объекта Socket
public InetAddress() public getLocalAddress()	Возвращает объект InetAddress, содержащий локальный адрес, с которым соединяется объект Socket
public int getPort()	Возвращает удаленный порт, с которым соединяется объект Socket
public int getLocalPort()	Возвращает локальный порт, с которым соединяется объект Socket
public OutputStream() public getOutputStream()	Возвращает выходной поток объекта Socke
void close()	Закрывает объект Socket
public string toString()	Возвращает IP-адрес и номер порта сокета клиента как String

ServerSocket представляет собой класс, используемый программами сервера для прослушивания запросов клиентов. ServerSocket реально не выполняет

сервис, но создает объект Socket от имени клиента, через который выполняется взаимодействие с сокетом клиента.

Для создания и инициализации объектов ServerSocket используются следующие конструкторы, определенные в классе ServerSocket:

– public ServerSocket(int port_number)- создает сокет сервера на заданный порт на локальной машине. Клиентам следуетиспользовать этот порт, чтобы общаться с сервером. Если номер порта 0, то сокет сервера создается на любой свободный порт локальной машины;
– public ServerSocket(int port, int backlog) - создает сокет сервера на заданный порт на локальной машине. Второй параметр задает максимальное количество соединений клиентов, которые сокет сервера поддерживает на заданном порту;
– public ServerSocket(int port, int backlog, InetAddress bindAddr) - создает сокет сервера на заданный порт. Третий параметр используется для создания сокета сервера хоста, подключенного к нескольким физическим линиям (multi-homed host). Сокет сервера принимает запросы клиента только с заданных IP адресов.

Некоторые полезные методы класса ServerSocket представлены в таблице 2.2.

Таблица 2.2- Основные методы класса ServerSocket

Метод	Описание
public InetAddress() public getInetAddress()	getInetAddress() Возвращает объект InetAddress, который содержит адрес объекта ServerSocket
public int getLocalPort()	Возвращает номер порта, с которого объект ServerSocket слушает запросы клиента

public Socket accept() throws IOException	Заставляет сокет сервера слушать соединение клиента и принимать его. После установления соединения клиента с сервером метод возвращает сокет клиента
public void bind(SocketAddress address) throws IOException	Связывает объект ServerSocket с заданным адресом (IP адрес и порт). Этот метод вызывает исключительную ситуацию IOException, когда происходит ошибка
public void close() throws IOException	Закрывает объект ServerSocket. Этот метод вызывает исключительную ситуацию IOException, когда происходит ошибка
public String toString()	Возвращает IP адрес и номер порта сокета сервера как String

2.2.2.3 Создание северной части по протоколу TCP/IP в Java

Для создания серверной системы сокета TCP, необходимо выполнить следующие шаги:

– создать объект сокета сервера ServerSocket (класс Server);
– прослушивать запросы клиента на соединение;
– запустить сервер;
– создать поток соединения для запросов клиентов.

2.2.2.4 Разработка класса Server

Класс Server наследует класс Thread и, таким образом, поддерживает многопоточное выполнение. Объект ServerSocket "слушает" запросы клиентов и конструктор класса Server создает объект ServerSocket. Сообщение об ошибке отображается, если возникает исключительная ситуация при запуске сервера.

Фрагмент кода для конструктора сервера показан в листинге 2.2

Листинг 2.2- Фрагмент конструктора класса Server

```
public Server()
{
```

```
 try
 {
 serverSocket = new ServerSocket(1001);
 }
 catch(IOException e)
 {
 fail(e, "Невозможно запустить сервер.");
 }
 System.out.println("Сервер запущен. . .");
 this.start(); // Запускается поток
}
```

В приведенном фрагменте кода используется общий метод обработки ошибок fail(), который обеспечивает обработку всех исключительных ситуаций. Метод принимает два аргумента (объект Exception и объект String) и выводит сообщение об ошибке.

Фрагмент кода для метода fail() выглядит следующим образом (листинг 2.3):

Листинг 2.3- Формирование сообщение об ошибке

```
public static void fail(Exception e, String str)
{
 System.out.println(str + "." + e);
}
```

2.2.2.5 «Прослушивание» запросов клиентов

Метод run() класса Server, как любой поток, который реализует интерфейс Runnable, содержит инструкции для потока. В этом случае сервер переходит в бесконечный цикл и прослушивает запросы клиентов. Когда сервер обнаруживает подключение клиента, метод accept() класса ServerSocket выполняет соединение. При этом сервер создает объект класса Connection для клиента. Объект класса Socket передается конструктору класса Connection и взаимодействие между клиентом и сервером выполняется через этот сокет.

Фрагмент кода для метода run() выглядит показан на листинге 2.4.

Листинг 2.4- Метод для прослушивания запросов клиентов

```
public void run()
{
 try
 {
 while(true)
 {
 Socket client = serverSocket.accept();
 Connection con = new Connection(client);
 }
 }
 catch(IOException e)
 {
 fail(e, “Не прослушивается”);
 }
}
```

2.2.2.6 Запуск сервера

Фрагмент кода для метода main() приведен в листинге 2.5.

Листинг 2.5- Запуск сервера

```
public static void main(String args[])
{
 new Server();
}
```

В этом фрагменте кода создается объект класса Server, который запускает поток.

2.2.2.7 Создание потокового соединения

Класс Connection нужен для потокового соединения с клиентом (листинг 2.6).

Листинг 2.6- Класс для создания потокового соединения

```
class Connection extends Thread
{
protected Socket netClient;
 protected BufferedReader fromClient;
 protected PrintStream toClient;
 public Connection(Socket client)
 {
 netClient = client;
 try
 {
fromClient = new BufferedReader(new
InputStreamReader(netClient.getInputStream()));
toClient = new PrintStream(netClient.getOutputStream());
 }
 catch(IOException e)
 {
 try
 {
 netClient.close();
 }
 catch(IOException e1)
 {
 System.err.println("Unable to set up streams"
+ e1);
 return;
 }
 }
 this.start();
 }
 public void run()
 {
 String clientMessage;
 try
 {
```

```
for(;;)
{
clientMessage = fromClient.readLine();
if(clientMessage == null)
break;
// Посылает подтверждение клиенту
toClient.println("Received");
}
}
catch(IOException e)
{}
finally
{
try
{
netClient.close();
}
catch(IOException e)
{}
}
}
}
```

В представленном фрагменте кода класс Connection создает объект fromClient класса BufferedReader, который получает ввод от клиента, используя метод getInputStream(). Объект класса PrintStream (toClient) дает возможность серверу отправлять клиенту данные, используя метод getOutputStream(). Таким образом, возникают необходимые (прием и передача) возможности взаимодействия.

Когда клиент соединяется с сервером, сервер использует метод readLine() объекта fromClient, чтобы запомнить сообщение, посланное клиентом в переменной clientMessage типа String. Метод println() используется для вывода сообщения “Received ” сокету.

Для выхода из системы сервер завершает цикл. При этом выполняется блок finally для закрытия сокета клиента. Закрытие сокета является важным действием, поскольку сохранение активного соединения неизбежно приводит к потери памяти сервера. Блок finally обеспечивает закрытие ранее установленного соединения. Следует отметить, что сервер является многопоточным, и каждый клиент получает свой собственный поток на сервере.

2.2.2.8 Пример создания класса Server

В листинге 2.7 показан создания класса Server, который принимает запросы соединения клиента и посылает строку Login, как ответное сообщение клиенту.

Листинг 2.7- Пример кода класса Server

```
import java.io.*;
import java.net.*;
public class Server extends Thread {
ServerSocket serverSocket; // Определяется переменная serverSocket
public Server() {
try {
/*
* Создание объекта ServerSocket, который принимает запросы
* соединения от клиентов через порт 1001
*/
serverSocket = new ServerSocket(1001);
System.out.println(serverSocket.toString());
} catch (IOException e) {
fail(e, "Could not start server.");
}
System.out.println("Сервер запущен . . .");
/* Стартует поток */
this.start();
}
```

```
public static void fail(Exception e, String str) {
System.out.println(str + "." + e);
}
public void run() {
try {
while (true) {
/* Принимаются запросы от клиентов */
Socket client = serverSocket.accept();
/*
 * Создается объект соединение, чтобы управлять
взаимодействием
* кдиента с сервером
*/
Connection con = new Connection(client);
}
} catch (IOException e) {
fail(e, "Not listening");
}
}
public static void main(String args[]) {
/* Запускается сервер */
new Server();
}
}
class Connection extends Thread {
/* Declare the variables */
protected Socket netClient;
protected BufferedReader fromClient;
protected PrintStream toClient;
public Connection(Socket client) {
netClient = client;
try {
/* Создается входной поток, чтобы принимать данные от
клиента */
fromClient = new BufferedReader(new
```

```
InputStreamReader(
netClient.getInputStream()));
/* Создается выходной поток, чтобы посылать данные
клиенту */
toClient = new
PrintStream(netClient.getOutputStream());
} catch (IOException e) {
try {
/* Закрывается сокет клиента */
netClient.close();
} catch (IOException e1) {
System.err.println("Unable to set up streams" +
e1);
return;
}
}
/* Start the thread */
this.start();
}
public void run() {
String login, password;
try {
while (true) {
toClient.println("Login: ");
/* Принимается login как ввод от клиента */
login = fromClient.readLine();
/* Завершить соединение, когда Bye вводится как login */
if (login == null || login.equals("Bye")) {
System.out.println("Exit");
return;
} else
System.out.println(login + " logged
in");
// Посылается подтверждение клиенту
toClient.println("Password: ");
```

```
/* Принимается пароль то клиента */
password = fromClient.readLine();
}
} catch (IOException e) {
} finally {
try {
netClient.close();
} catch (IOException e) {
}
}
}
}
```

2.2.2.9 Создание клиента TCP/IP

Первый шаг выполнения клиента состоит в установлении соединения между клиентом и сервером. Для установления соединения между клиентом и сервером необходимо создать объект Socket. Для создания приложения клиента сокета TCP необходимо выполнить следующие задачи:

– создать сокет клиента, используя объект Socket;
– читать и писать в сокет;
– закрыть соединение.

2.2.2.10 Создание сокета клиента

Объект сокета клиента создается с помощью конструктора класса Socket, принимающего два параметра, IP адрес и номер порта, которые прослушиваются сервером. Следующий фрагмент кода используется для создания сокета клиента (листинг 2.8).

Листинг 2.8- Создание сокета клиента

```
try
{
 Socket clientSocket = new Socket("127.0.0.1", 1001);
```

```
}
catch(UnknownHostException e)
{
System.err.println("Неопределенное имя хоста ");
System.exit(1);
}
```

В предыдущем фрагменте кода IP адрес равный ‘127.0.0.1’ и порт равный ‘1001’ определяют сокет, на котором сервер прослушивает запросы клиента.

2.2.2.11 Чтение и запись в сокет

После установления соединения между клиентом и сервером, клиент посылает запрос на сервер через сокет. Чтение и запись в сокет аналогичны чтению из файла и записи в файл. Чтобы обеспечить возможность клиенту общаться с сервером, необходимо выполнить следующие действия:

– объявляются два объекта по одному для классов PrintStream и BufferedReader. Эти объекты будут использоваться для чтения и записи в сокет socket. PrintStream out = null; // Объект для записи в сокет BufferedReader in = null; // Объект для чтения из сокета;
– объекты PrintStream и BufferedReader связываются с сокетом. out = new PrintStream(clientSocket.getOutputStream()); in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

Методы getInputStream() и getOutputStream() класса Socket позволяют клиенту взаимодействовать с сервером. Метод getInputStream() позволяет объекту BufferedReader читать из сокета, а метод getOutputStream() позволяет объекту PrintStream писать в сокет.

Объявляется еще один объект класса BufferdReader для связи со стандартным входом, чтобы данные, введенные в приложении клиенте, могли передаваться на сервер. Следующий фрагмент кода используется для чтения данных из окна консоли (листинг 2.9).

Листинг 2.9- Чтение данных из окна консоли

```
BufferdReader stdin = new BufferedReader(new
InputStreamReader(System.in));
String str;
while((str = stdin.readLine()).length() != 0)
{
out.println(str);
}
```

Представленный фрагмент кода позволяет пользователю вводить данные с клавиатуры. Цикл while продолжается до тех пор, пока пользователь не введет символ завершения ввода (Ctrl-Z).

2.2.2.12 Закрытие соединения

Операторы, представленные ниже, закрывают потоки и соединения с сервером: out.close(); in.close(); stdin.close().

Сокет клиента принимает имя пользователя и пароль, и обеспечивает связь. Для завершения соединения, пользователь должен ввести ‘Bye’. Можно использовать следующий код для создания класса Client (листинг 2.10).

Листинг 2.10- Пример создания класса Client

```
import java.net.*;
import java.io.*;
public class Client {
public static void main(String[] args) throws IOException {
Socket clientSocket;
PrintStream out = null;
BufferedReader in = null;
try {
/* Создается объект сокет, чтобы соединиться с сервером
*/
clientSocket = new Socket("127.0.0.1", 1001);
/* Создается выходной поток, чтобы посылать данные на
```

```
сервер */
out = new
PrintStream(clientSocket.getOutputStream());
/* Создается входной поток, чтобы принимать данные с
сервера */
in = new BufferedReader(new InputStreamReader(
clientSocket.getInputStream()));
} catch (UnknownHostException e) {
System.err.println("Unidentified hostname ");
System.exit(1);
} catch (IOException e) {
System.err.println("Couldn't get I/O ");
System.exit(1);
}
/* Создается входной поток, чтобы читать данные из окна консоли
*/
BufferedReader stdin = new BufferedReader(new
InputStreamReader(
(System.in)));
/* Чтение из сокета */
String login = in.readLine();
System.out.println(login);
/* Прием login */
String logName = stdin.readLine();
out.println(logName);
/* Чтение из сокета */
String password = in.readLine();
System.out.println(password);
/* Прием password */
String pass = stdin.readLine();
out.println(pass);
String str = in.readLine();
System.out.println(str);
while ((str = stdin.readLine()) != null) {
out.println(str);
```

```
if (str.equals("Bye"))
break;
}
out.close();
in.close();
stdin.close();
}
}
```

Представленный выше код сохраняется как Client.java. При выполнении класса Client, отображается сообщение для ввода Login. После приема Login на консоли класса Client появляется сообщение для ввода password. Рис. 7 показывает вывод класса Client, который посылает login XXXX и password для входа на Server. Введенный login в окне консоли класса Client передается классу Server. Когда на сервер передается строка «Bye» в качестве login от клиента, соединение между клиентом и сервером прекращается.

2.2.3 Порядок выполнения задач ПЗ

Алгоритм выполнения задач практического занятия следующий:

- выбрать 5 задач по следующему правилу: номер по журналу- первая задача; номер каждой последующей задачи определяется прибавлением цифры 3 к номеру первой задачи, который только что вычислили (если достигнуто окончание списка вариантов здач, то перейти в его начало);
- составить классы Swrver и Client и установить соединение между ними;
- составить методы (функции) решения всех задач, поместить их в класс Server. Все исходные данные вводить в консоли клиента, передавать их на сервер, а полученные результаты решений задач выводить на экран в консоли клиента;
- оформить отчет для всей системы в целом, включив в него задание, блок-схему алгоритма (в электронном виде), текст программы и skrin-shert результата выполнения каждой задачи и представить его на проверку.

2.2.4 Варианты заданий:

1. Составить программу для перевода длины в метрах в длину в сантиметрах, определив функцию, выполняющую это преобразование и передав длину в метрах в качестве параметра.
2. Составить программу для нахождения суммы элементов каждого из трех массивов, введенных с клавиатуры, определив функцию, выполняющую это действие, и передавая массивы в качестве параметра.
3. Даны числа S, T. Получить с использованием функции пользователя F(T,-2S,1.17)+F(2.2,T,S-T) где F(A, B, C) = (2A-B-sin(C))/(5+C)
4. Составить программу перевода двоичной записи натурального числа в десятичную, описав соответствующую функцию с параметром. Перевод осуществлять для чисел, вводимых с клавиатуры. Признак конца ввода - число 0.
5. Даны числа S, T. Получить с использованием функции пользователя с параметрами G(1,sin(S))+2G(T*S,24)-G(5,-S), где G(A,B)=(2A+B*B)/(A*B*2+B*5).
6. Составить программу для расчета значений гипотенузы треугольника, определив функцию, выполняющую этот расчет. Катеты передаются в качестве параметров.
7. Найти периметр десятиугольника, координаты вершин которого заданы. Определить процедуру вычисления расстояния между двумя точками, заданными своими координатами, которые передаются функции в качестве параметров из основной программы.
8. Найти периметр шестиугольника, координаты вершин которого заданы. Определить процедуру вычисления расстояния между двумя точками, заданными своими координатами. Координаты передаются функции в качестве параметров из основной программы.
9. Найти площадь пятиугольника, координаты вершин которого заданы. Определить процедуру вычисления расстояния между двумя точками, заданными своими координатами, и процедуру вычисления площади тре-

угольника по трем сторонам. Описать функции с соответствующими формальными параметрами.

10. Составить программу вывода на экран всех натуральных чисел, не превосходящих N и делящихся на каждую из своих цифр. Описать соответствующую функцию, получающую из основной программы в качестве параметра натуральное число и возвращающую TRUE, если оно удовлетворяет указанному условию.
11. Используя подпрограмму - функцию, составить программу для нахождения максимального из трех чисел. Числа передаются функции в качестве параметров.
12. Используя подпрограмму - функцию, составить программу для печати знаков трех чисел, введенных с клавиатуры и передаваемых функции в качестве параметра.
13. Используя подпрограмму - функцию, составить программу для возведения чисел в целую положительную степень. Число передаются функции в качестве параметра из основной программы. Расчет вести для чисел, пока не будет введено число, равное 0.
14. Используя подпрограмму - функцию, составить программу для вычисления функции Z=(X1+Y1)/(X1*Y1), где X1 - первый корень уравнения X^2-4*X-1=0; Y1 - первый корень уравнения $2*Y^2 + A*Y - A^2 = 0$ (A - произвольное).
15. Задав функцию, вывести на печать средние арифметические двух массивов, введенных с клавиатуры. Массив передается функции в качестве параметра.
16. Задав функцию, рассчитать и вывести на печать максимальные значения в трех парах чисел, вводимых с клавиатуры. Пара чисел передается функции в качестве параметра.
17. Найти периметр восьмиугольника, координаты вершин которого заданы. Определить функцию вычисления расстояния между двумя точками, заданными своими координатами. Координаты передать функции в качестве параметров.

18. Даны четыре пары чисел. Получить с использованием функции пользователя наибольший общий делитель для каждой пары.
19. Даны числа A, B, C. Получить с использованием функции пользователя наименьшее значение. Числа передаются функции из основной программы в качестве параметров.
20. Даны числа x = 1,2,...,N. Получить с использованием функции пользователя значения 3*P(X+3)*P(X) для заданных x, где $P(X) = 10*X^3 - 14*X^2 + 12*X - 2$.
21. Составить программу для расчета значений катета треугольника, определив функцию, выполняющую этот расчет. Гипотенуза и второй катет передаются в качестве параметров.
22. Даны целые числа a,b,c,d. Проверить с использованием функции пользователя их четность. Число для проверки передается в функцию в качестве параметра из основной программы.
23. Для каждого из 10 введенных с клавиатуры чисел напечатать сообщение: является ли оно простым или нет, описав функцию логического типа, возвращающую значение “ИСТИНА”, если число, переданное ей в качестве параметра, является простым.
24. Даны числа S, T. Получить с использованием функции пользователя Y(T,S)=G(12,S)+G(T,S)-G(2S-1,S*T), где G(A,B)=(2*A+B*B)/(A*B*2+B*5).
25. Определите функцию, определяющую, какой целой степенью числа 2 является ее аргумент (если число не является степенью двойки - выдать соответствующее сообщение).
26. Определите функцию, подсчитывающую сумму N первых элементов целочисленного массива A. N и массив A передать в качестве параметров.
27. Вычислить количество простых чисел, не превосходящих заданного N. Описать функцию логического типа, возвращающую значение true, если число простое и false в противном случае.

28. Используя подпрограмму - функцию с параметрами, составить программу для вычисления функции $F(X,Y) = (2X^3-4*X^2+X+1)/(9*Y^3+Y+4) + 3*Y^2+5*Y$.
29. Составить программу для перевода веса в граммах в вес в килограммах, определив функцию, выполняющую это преобразование. Вес в граммах передается функции в качестве параметра.
30. Даны числа S, T. Получить с использованием функции пользователя G(12, S)+G(T, S)-G(2S-1, S*T) где G(A, B) = (2*A+B*B)/(A*B*2+B*5).

2.3 ПЗ-3. Простые двухуровневые клиент-серверные приложения в C#

2.3.1 Цели занятия

Выработать умения и навыки составлять типовые программы решения задач на выбранном языке программирования, снабженные элементами графического интерфейса пользователя в виде клиент-серверных систем.

2.3.2 Теоретические основы и пример выполнения

Для создания клиент-серверного приложения в C#, аналогично предыдущему ПЗ, необходимо создать два класса Server и Client, используя технологию сокетов.

2.3.2.1 Программирование класса Server

Вначале в этом классе создадим серверное соединение и используем для этого систему сокетов.

Сокеты − это концепция сетевого программирования, когда существуют два вида «программных разъемов»- сокетов: клиентские и серверные.

Серверный сокет как розетка, которая «висит на стене» готовая к работе, в ожидании, когда к ней подключат штекер. Точно так же серверный сокет пере-ходит в режим ожидания подключения на определенном адресе и определенном порту.

Клиентский сокет как вилка, которую втыкают в розетку. Как только клиент подключается к серверному сокету, информация начинает передаваться между ними.

Для того чтобы создать виртуальное подключение между клиентом и сервером, надо знать место, где находится нужный нам серверный сокет.

В нашем случае, создается слушатель событий подключений в сети по протоколу TCP IP – TcpListener, в качестве параметра которому передается так называемая точка входа, которую определяет метод IPEndPoint из переданных ему IP адреса компьютера, на котором будет запущен сервер (пока это вариант localhost, т.е. IP 127.0.0.1) и номер порта Port, на котором создаваемой серверное подключение будет прослушивать подключения клиентов. Т.о., в конструкторе класса Server появится следующий слушатель (листинг 2.11).

Листинг 2.11- Серверный слушатель подключения клиентов

```
TcpListener listner;
        public Program(int Port)
        {
            listner = new TcpListener(new IPEnd-
Point(IPAddress.Parse("127.0.0.1"), Port));
            listner.Start();
            Console.WriteLine("Сервер запущен, ожидает подклю-
чений...");
. . . }
```

Видно, что после запуска прослушивания сетевых подключений выдается сообщение о том, что сервер готов и ожидает подключений.

Далее, нужно в это подключение с помощью метода AcceptTcpClient() и создать новый поток для него с помощью метода Thread из одноименной библиотеки, которую нужно подключить заранее: using System.Threading.

В качестве параметра этому методу передается метод clientThread, в котором для каждого нового клиента открывается TCP соединение в новом потоке (листинг 2.12).

Листинг 2.12- Метод создания нового TCP соединения для нового клиента

```
static void clientThread(Object StateInfo)
        {
           new Client((TcpClient)StateInfo);
        }
```

В итоге получим следующий текст класса Server (листинг 2.13).

Листинг 2.13- Класс Server

```
class Server
    {
       TcpListener listner;
        public Server (int Port)
        {
            listner = new TcpListener(new IPEnd-
Point(IPAddress.Parse("127.0.0.1"), Port));
            listner.Start();
            Console.WriteLine("Сервер запущен, ожидает подклю-
чений...");
            while (true)
            {

                TcpClient client = listner.AcceptTcpClient();
                // Создаем поток


  Thread Thread = new Thread(new ParameterizedThread-
Start(clientThread));
                // И запускаем этот поток, передавая ему при-
нятого клиента
                Thread.Start(client);
            }
        }
```

```
        static void clientThread(Object StateInfo)
        {
           new Client((TcpClient)StateInfo);
        }

        static void Main(string[] args)
        {
            new Program(12000);
        }
    }
```

Видно, что сюда же включен и главный метод модуля – метод Main, из которого происходит запуск всего приложения и сервер начинает «прослушивание» на порту №12000, который в качестве параметра передается конструктору класса.

2.3.2.2 Класс Client

Этот класс необходим для считывания запроса от клиентских приложений, поиска нужного контента согласно запросу и для его передачи в сеть клиентам.

Начнем с задания объектов для методов чтения из потока и записи в поток. Для этого создаются новые экземпляры соответствующих классов (листинг 2.14).

Листинг 2.14- Организация потокового чтения и записи

```
StreamReader sr = new StreamReader(client.GetStream()); // чтение из потока клиента

            StreamWriter sw = new StreamWriter(client.GetStream()); // запись в поток клиента

            sw.AutoFlush = true;
```

Этот пример клиент-серверного приложения обеспечивает запрос от клиента на сервер необходимого файла информации, передачу его клиенту, сохранение в темповском файле на клиенте и отображение в клиентской части приложения. Поэтому, определим какой файл необходим: прочитаем из потока в за-просе клиента значение строковой переменной str, конвертируем его в целочисленную переменную kluch, и по ее значению в структуре switch определим необходимый файл (листинг 2.15).

Листинг 2.15- Определение требуемого контента

```
// Принимаем передачу от клиента
            string cl = sr.ReadLine(); // Отображение типа обучающего модуля
            Console.WriteLine(cl);
            string str = sr.ReadLine();
            int kluch = int.Parse(str); // по kluch определяем нужный файл
            string FilePath = "C:/www/"; // составляем путь к файлу
            switch (kluch)
            {
                case 1:
                    {
                        FilePath += "L1.htm";
                        Console.WriteLine("Путь к файлу: " + FilePath);
                    }
                    break;
```

```
                case 2:
                    {
                        FilePath += "L2.htm";
                        Console.WriteLine("Путь к файлу: " +
FilePath);
                    }
                    break;
            }
```

Видно, что весь контент преобразован в html формат и находится в соответствующих папках на сервере. Аналогично программируется определение оставшихся файлов обучающих материалов.

Далее нужно открыть найденный файл. Сделаем это, проверяя правильность его открытия с помощью блока try/catch и если возникнет ошибка открытия, выдадим соответствующее сообщение (листинг 2.16).

Листинг 2.16- Безопасное открытие файла контента

```
// Открываем файл, страхуясь на случай ошибки
            FileStream FS;
            try
            {
                FS = new FileStream(FilePath, FileMode.Open, File-
Access.Read, FileShare.Read);
                sw.WriteLine("Запрашиваемый файл открыт и го-тов к
передаче");
            }
```

```
            catch (Exception)
            {
                // Если случилась ошибка, посылаем клиенту ошибку открытия файла
                sw.WriteLine("Ошибка открытия файла");
                return;
            }
```

Учитывая, что данные передаются в сети в формате байт, нужно зарезервировать буфер для чтения материалов из открытого файла. Затем, перенести эти данные в буфер и через него передать их клиенту в ответе на запрос. Весь этот процесс будет выполняться до конца данных в открытом файле (листинг 2.17).

Листинг 2.17- Передача контента в ответном сообщении клиенту

```
// Задаем буфер для размещения данных из выбранного файла
            byte[] Buffer = new byte[1024];
            // Переменная для хранения количества байт, передаваемых клиенту
            int Count;

            // Пока не достигнут конец файла
            while (FS.Position < FS.Length)
            {
                // Читаем данные из файла и пишем их в буфер
                Count = FS.Read(Buffer, 0, Buffer.Length);
                // И передаем их клиенту
```

```
            client.GetStream().Write(Buffer, 0, Count);
        }
        sw.WriteLine("Файл передан");
        // Закроем файл и соединение
        FS.Close();
        sw.WriteLine("Соединение прервано");
        client.Close();
```

Из листинга следует, что по завершению передачи данных клиенту мы закрываем файл и соединение.

2.3.3 Порядок выполнения задач ПЗ

Алгоритм выполнения задач практического занятия следующий:

- выбрать 5 задач по следующему правилу: номер по журналу- первая задача; номер каждой последующей задачи определяется прибавлением цифры 3 к номеру первой задачи, который только что вычислили (если достигнуто окончание списка вариантов здач, то перейти в его начало);
- составить классы Swrver и Client и установить соединение между ними;
- составить методы (функции) решения всех задач, поместить их в класс Server. Все исходные данные вводить в консоли клиента, передавать их на сервер, а полученные результаты решений задач выводить на экран в консоли клиента;
- оформить отчет для всего приложения вцелом, включив в него задание, блок-схему алгоритма (в электронном виде), текст программы и skrin-shert результата выполнения каждой задачи и представить его на проверку.

2.3.4 Варианты заданий

1. 1. Вывести на печать положительные значения функции y=sin(x)+5cos(x-2) для x изменяющегося на отрезке [-5, 12] с шагом 1,2.
2. Вывести на печать значения функции z=tg(2x)-sin(x) для x изменяющегося на отрезке [-3, 3] с шагом 0,3.
3. Ввести с клавиатуры и напечатать модули N чисел; если введено отрицательное число, ввод и печать прекратить.
4. 4. Вывести на печать значения функции z=ln(x)+tg(2x), большие 1, для x изменяющегося на отрезке [3, 8] с шагом 0,9.
5. Определить, является ли натуральное число N степенью числа 5 или нет.
6. Напечатать значения функции y=ln(x+12/x),где значения x вводятся с клавиатуры. При вводе числа, не входящего в область определения функции, вычисления прекратить.
7. Напечатать значения функции y=ln(x-1/x),где значения x вводятся с клавиатуры. При вводе числа,не входящего в область определения функции, вычисления прекратить.
8. Для x из интервала (-2;8) с шагом 0,75 вычислить y=(4x-3x+tg(x))/A, где A вводится с клавиатуры.
9. Вывести на печать значения функции z=sin(x)+cos(x),находящиеся в интервале (-0,2; 0,8) для x изменяющегося на отрезке [4,-6] с шагом 0,91.
10. Дано натуральное число N. Получить наименьшее число вида 4^k, большее N.
11. Для x из интервала (2;8) с шагом 0,75 вычислить y=(4x-3x+cos(x))/A, где A вводится с клавиатуры.
12. Найти первый член последовательности ln(9n)/(n*n), меньший 1, для n изменяющегося следующим образом: n=1,2,3... .
13. Определить, является ли натуральное число N степенью числа 3 или нет.
14. 14. Вывести на печать отрицательные значения функции z=cos(x)-5sin(x-2) для x изменяющегося на отрезке [-3, 11] с шагом 0,9.

15. Ввести с клавиатуры и напечатать квадраты N чисел, если введено кратное 3 положительное число, ввод и печать прекратить.
16. Вывести на печать отрицательные значения функции z=tg(x)+5cos(x-2) для x изменяющегося на отрезке [12, 1] с шагом 1,2.
17. Ввести с клавиатуры и напечатать N чисел, если введено равное нулю или кратное 2 число, ввод и печать прекратить.
18. Вывести на печать значения функции z=ln(|x|)+tg(2x), большие 2 для x изменяющегося на отрезке [3, -8] с шагом 0,9.
19. Найти первый отрицательный член последовательности sin(tg(n/2)) для n изменяющегося на следующим образом: n=1,2,3... .
20. Напечатать значения функции y=ln(x+12/x),где значения x вводятся с клавиатуры. При вводе числа, не входящего в область определения функции, вычисления прекратить.
21. Найти первую цифру в целом положительном числе.
22. Дано натуральное число N. Получить наибольшее число вида 3^k, меньшее N.
23. Вывести на печать значения функции z=sin(x)+cos(x),находящиеся в интервале (-0,3;0,7) для x изменяющегося на отрезке [-4,6] с шагом 0,91.
24. Дано натуральное число N. Получить наименьшее число вида 5^k, большее N.
25. Для x из интервала (-2;8) с шагом 0,75 вычислить y=(4x-3x+tg(x))/A, где A вводится с клавиатуры.
26. Найти первый член последовательности ln(9n/(n*n+1), меньший 0, для n изменяющегося на следующим образом: n=1,2,3... .
27. Определить, является ли натуральное N степенью числа 4 или нет.
28. Вывести на печать положительные значения функции z=sin(x)-5cos(x-2) для x изменяющегося на отрезке [5,-12] с шагом 1,2.

29. Напечатать значения функции $Y = \sqrt{2x^2 - x^3}$ для произвольных *x,* вводимых с клавиатуры. При вводе числа, не входящего в область определения функции, ввод и печать прекратить.
30. Найти первый отрицательный член последовательности cos(ctg(n)) для n изменяющегося на следующим образом: n=1,2,3... .

ЗАКЛЮЧЕНИЕ

В учебном пособии подробно и наглядно изложены основы выполнения заданий практических занятий по дисциплине «Проектирование клиент-серверных систем».

Материал пособия снабжен необходимыми примерами выполнения типовых задач разработки клиент-серверных систем на языках Java и C# для интегрированных сред разработки Eclipse и Visual Studio соответственно. Здесь же приведены необходимые банки заданий для проведения указанных практических занятий.

Пособие соответствует требованиям, предъявляемым к учебно-методическому комплексу указанной дисциплины, обсуждено на заседании кафедры ИВТ и рекомендовано к использованию в учебном процессе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дубаков А.А. Сетевое программирование: учебное пособие / А.А. Дубков – СП: НИУ ИТМО, 2013. – 248 с.
2. Грамотная клиент-серверная архитектура: как правильно проектировать и разрабатывать web API. [Электронный ресурс] // [сайт] [2018], URL: https://tproger.ru/articles/web-api/. (дата обращения: 03.05.2018).
3. Технологии разработки клиент-серверных приложений. [Электронный ресурс] // [сайт] [2018], URL: https://bourabai.ru/dbt/client2.htm. (дата обращения: 05.05.2018).
4. Сенина А.А., Тузовский А.Ф Обзор основных современных технологий разработки web-приложений. [Электронный ресурс] // [сайт] [2018], URL: https://core.ac.uk/download/pdf/53095773.pdf. (дата обращения: 03.05.2018).
5. Басс Лен, Клементс Пол, Кацман Рик. Архитектура программного обеспечения на практике. 2-е издание. — СПб.: Питер, 2006. — 575 с.
6. П. Дейтел, Х. Дейтел, Э. Дейтел, М. Моргано. Android для программистов: создаём приложения. — СПб.: Питер, 2013. — 560 с.
7. Краткое описание Visual Studio [Электронный ресурс] // [сайт] [2018], URL: http://habrahabr.ru. (дата обращения: 10.05.2018).
8. Фленов М. Е. Библия C#. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2011. — 560 с.: