Министерство образования Российской Федерации
Томский политехнический университет

Факультет АВТ
Кафедра ОСУ

Курсовая работа
“Клиент для сетевого чата”

Выполнил:
студент гр. 8В15
/ Дзалбо В.В. /

Проверил преподаватель:
/ Кузнецов Д. Ю. /

Томск 2004

Министерство образования РФ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра_____

УТВЕРЖДАЮ:
Зав. кафедрой_____Силич
(Подпись, дата)

ЗАДАНИЕ
на выполнение
(курсового проекта, работы)

Студенту_____

1.Тема курсового проекта Клиент для сетевого чата
2.Срок сдачи студентом готовой работы 8 июня 2004 г.
3. Содержание текстового документа (перечень подлежащих разработке вопросов)

_____

_____

_____

4.Перечень графического материала (с точным указанием обязательных чертежей)
5.Консультанты по разделам курсовой работы
6.1.
_____
6.2.
_____
6.3.
_____
6.4.
_____

7.Дата выдачи задания на выполнение курсового проекта, работы
_____

Руководитель_____
(подпись, дата)

Задание принял к исполнению

_____
(подпись, дата)

Содержание

1. **Введение**

Данная курсовая работа является примером реализации клиента сетевого чата, предназначенного для обмена сообщениями в сети и работающего по принципу клиент-сервер. Клиент реализован с применением принципов Объектно-Ориентированного программирования на языке высокого уровня Java. Реализованная программа сетевого общения работает на базе протокола TCP посредством которого происходит подключение клиента к серверу. Программа-сервер, также реализованая на языке Java, использует простой формат текстовых сообщений, принимает их от клиента и передает всем другим подключенным к серверу клиентам. (Является упрощенным вариантом сервера передачи текстовых сообщений, использующегося при работе http://hbz.tomsk.ru)

Объектно-ориентированное программирование (ООП) — это методика, которая концентрирует основное внимание программиста на связях между объектами, а не на деталях их реализации. Объект - это абстрактная сущность, наделенная характеристиками объектов окружающего нас реального мира.

ОО Методология – методология, основанная на объектно-ориентированном подходе. Цель создания – устранение недостатков структурной методологии.

Система описывается как совокупность взаимодействующих объектов, принадлежащих некоторым классам, классы образуют иерархию наследования. (Г. Буч, ОО анализ и проектирование)

Три базовых понятия ООП

Инкапсуляция.

Сокрытие данных. Объединение данных и методов работы с ними. Смысл понятия в том что, каждый элемент системы является объектом.

Наследование.

Классы образуют иерархию. Суперкласс (предок) нескольких подклассов (потомков) имеет интерфейс, являющийся подмножеством пересечения интерфейсов подклассов. Условие правильности суперкласса является

следствием условия для любого подкласса. По смыслу суперкласс – обобщение своих подклассов. Подкласс – уточнение суперкласса.

Откуда берется наследование:

1. Обобщение – например, List и Choice являются подклассами Component.
2. Роли – один объект может исполнять несколько ролей. Например, подкласс Component может быть подклассом DragSource, если ее можно тащить и подклассом DropTarget, если в нее можно кидать.

Полиморфизм.

Любой объект класса 1, реализующего интерфейс класса 2 может быть использован как объект класса 2.

Язык Java в полной мере яфляется объектно-ориентированным. Его ключевые свойства:

1. Классы.
2. Защищенные (private) поля.
3. Наследование.

Виртуальные методы (Виртуальный метод – это метод, связанный с первичным классом объекта, а не с классом использования объекта. Таким образом, при приведении типа объекта к типу предка он сохраняет свои виртуальные методы).

Объектно-ориентированный подход имеет много плюсов.

1. ОО описание системы акцентирует внимание на субъектах. Что приближает его к описанию на естественном языке. Любое действие (сказуемое) имеет явного или неявного субъекта (действующее лицо).
2. Динамическая и статическая проекции системы явно связаны через понятие объекта.
3. Присутствует контроль целостности потоков данных. Все потоки данных оперируют объектами.

Сам ОО подход не уменьшает явную структурную сложность системы. Он лишь дает возможность устранить часть неявных взаимосвязей между

подсистемами, а другую часть сделать явной.

Программа была полностью реализована на языке Java. Важнейшими свойствами этого языка, называемого даже революционным, являются следующие:

- Java является современным *объектно-ориентированным* языком;
- Java предоставляет программисту богатый набор *классов* различных объектов;

Выбор этого языка обусловлен прежде всего наличием у него таких качеств, как простота и мощь, безопасность, объектная ориентированность, надежность, интерактивность, архитектурная независимость, возможность интерпретации, высокая производительность и легкость в изучении. Объектно-ориентированное программирование настолько интегрировано в Java, что написание даже простейших программ требует знания основных принципов ООП.

Можно перечислить несколько основных проявлений ООП в языке Java.

- Класс состоит из данных и методов, работающих с данными.
- Объект является конкретным экземпляром класса.
- Доступ к членам объекта (переменным и методам) осуществляется с помощью оператора +точка¤, размещаемого между именем объекта и именем переменной.
- Копия переменных экземпляра (не статических) содержится в каждом объекте (экземпляре) класса.
- Статические переменные класса связаны с классом. Всегда существует ровно одна копия переменной класса независимо от числа экземпляров класса.
- Методам экземпляра (не статическим) неявно передается аргумент this, идентифицирующий объект, над которым производится действие.
- Статическим методам класса не передается аргумент this, и, следовательно, они не владеют текущим экземпляром класса, который можно использовать для неявной ссылки на переменные экземпляра

или для неявного вызова методов экземпляра.

- Объекты создаются при помощи ключевого слова new, которое вызывает конструктор класса с соответствующим списком аргументов.
- Объект не уничтожается явно. Сборщик мусора автоматически утилизирует объекты, которые больше не используются.
- Если в первой строке конструктор не вызывает другой конструктор при помощи this() или конструктор суперкласса при помощи super(), то автоматически вызывается конструктор суперкласса, который не имеет аргументов. В любом случае порождается цепочка вызовов конструкторов.
- Если конструктор в классе не определен, то автоматически создается конструктор по умолчанию.
- Класс может наследовать переменные и методы другого класса, не объявленные как private, становясь его подклассом, т.е. путем объявления другого класса в блоке extends.
- Класс java.lang.Object суперкласс любого класса по умолчанию. Это корневой класс иерархии классов в языке Java, не имеющий суперкласса. Все классы наследуют методы класса Object.
- Перегрузка методов определение множества методов, имеющих одинаковое имя, но разные списки аргументов.
- Переопределение методов возникает, когда класс переопределяет методы, унаследованные от его суперкласса.
- Динамический поиск методов гарантирует вызов корректного метода для объекта, даже если это экземпляр класса, в котором метод переопределен.
- Методы, объявленные как static, final, или private, не могут быть переопределены и не могут служить объектами динамического поиска. Это позволяет компилятору использовать inline-подстановку, оптимизируя их вызов.
- Из подклассов можно явно вызывать переопределенные методы

суперкласса с помощью ключевого слова super.

- Можно явно ссылаться на скрытые переменные с помощью ключевого слова super.
- Данные и методы могут быть скрыты и инкапсулированы внутри класса благодаря модификаторам видимости private и protected. Переменные класса, объявленные как public, видимы везде. Переменные класса без модификаторов видимости видны только внутри пакета.
- Абстрактный метод не имеет тела (т.е. реализации).
- Абстрактный класс содержит абстрактные методы. Методы должны быть реализованы в подклассе прежде, чем будут созданы экземпляры этого подкласса.
- Интерфейс в языке Java это набор абстрактных методов и констант (переменных, объявленных как final static). Объявление интерфейса создает новый тип данных.
- Класс реализует интерфейс, объявляя его в блоке implements и создавая тело метода для каждого из абстрактных методов интерфейса.

2. Проработка программных средств.

Весь проект реализован с использование Java и предоставляемых Java технологий и компонентов (swing, awt). Рассмотрим поподробнее, что из себя представляет этот язык программирования.

Основные характеристики Java

Развитие Internet и World Wide Web заставляет совершенно по-новому рассматривать процессы разработки и распределения программного обеспечения. Для того, чтобы выжить в мире электронного бизнеса и распространения данных, язык Java должен быть

- безопасным,
- высокопроизводительным,
- надежным.

Простота языка входит в ключевые характеристики Java: разработчик не должен длительное время изучать язык, прежде чем он сможет на нем программировать. Фундаментальные концепции языка Java быстро схватываются и программисты с самого начала могут вести продуктивную работу. Разработчиками Java было принято во внимание, что многие программисты хорошо знакомы с языком C++, поэтому Java, насколько это возможно, приближен к C++. .

В Java не включены некоторые редко используемые, плохо понимаемые и усложняющие работу возможности C++, которые приносят больше проблем, чем преимуществ. Пришлось отказаться от

- перегрузки операторов (но перегрузка методов в Java осталась),
- множественного наследования,
- автоматического расширяющего приведения типов.

Добавилась автоматическая сборка мусора, упрощающая процесс программирования, но несколько усложняющая систему в целом. В C и C++ управление памятью вызывало всегда массу проблем, теперь же об этом не придется много заботиться.

Платформа Java разработана для создания высоконадежного прикладного программного обеспечения. Большое внимание уделено проверке программ на этапе компиляции, за которой следует второй уровень - динамическая проверка (на этапе выполнения).

Модель управления памятью предельно проста: объекты создаются с помощью оператора new. В Java, в отличие от C++, механизм указателей исключает возможность прямой записи в память и порчи данных: при работе с указателями операции строго типизированы, отсутствуют арифметические операции над указателями. Работа с массивами находится под контролем управляющей системы. Существует автоматическая сборка мусора.

Данная модель управления памятью исключает целый класс ошибок, так часто возникающих у программистов на C и C++. Программы на Java можно писать, будучи уверенным в том, что машина не "повиснет" из-за ошибок при работе с динамически выделенной памятью.

Java разработана для оперирования в распределенных средах, это означает, что на первом плане должны стоять вопросы безопасности. Средства безопасности, встроенные в язык, и система исполнения Java позволяют создавать приложения, на которые невозможно "напасть" извне. В сетевых средах приложения, написанные на Java, защищены от вторжения неавторизованного кода, пытающегося внедрить вирус или разрушить файловую систему.

Java разработан для поддержки приложений, внедряемых в гетерогенные сетевые среды. В подобных средах приложения должны исполняться на различных аппаратных архитектурах, под управлением различных операционных систем и во взаимодействии с интерфейсами различных языков программирования. Для обеспечения платформо-независимости программ компилятор Java генерирует байт-код - архитектурно-нейтральный промежуточный формат программы, создаваемый для эффективной передачи кода на различные аппаратные и программные платформы. При выполнении программы байт-код интерпретируется исполняющей машиной Java. Один и

тот же Java-байткод будет исполняться на любой платформе.

Архитектурная независимость - лишь составная часть переносимости. В отличие от C или C++ в Java не существует понятия "зависимости от реализации", когда речь идет о размерности базовых типов. Форматы типов данных и операции над ними четко определены. Тем самым, программы остаются неизменными на любой платформе - не существует несовместимости типов данных на аппаратных и программных архитектурах.

Архитектурная независимость и переносимость программного обеспечения Java обеспечивается виртуальной машиной Java (Java Virtual Mashine - JVM) - абстрактной машиной, для которой компилятор Java генерирует код. Специальные реализации JVM для конкретных аппаратных и программных платформ предоставляют уже конкретную виртуальную машину. JVM базируется на стандарте интерфейса переносимых операционных систем (POSIX).

Производительность всегда заслуживает особого внимания. Java достигает высокой производительности благодаря специально оптимизированному байт-коду, легко переводимому в машинный код. Автоматическая сборка мусора выполняется как фоновый поток с низким приоритетом, обеспечивая высокую вероятность доступности требуемой памяти, что ведет к увеличению производительности. Приложения, требующие больших вычислительных ресурсов, могут быть спроектированы так, чтобы те части, которые требуют интенсивных вычислений, были написаны на языке ассемблера и взаимодействовали с Java платформой. В основном, пользователи ощущают, что приложения взаимодействуют быстро, несмотря на то, что они являются интерпретируемыми.

Java-интерпретатор может выполнять Java байт-код на любой машине, на которой установлен интерпретатор и система выполнения. На интерпретирующей платформе фаза сборки программы является простой и пошаговой, поэтому процесс разработки существенно ускоряется и упрощается, отсутствуют традиционные трудные этапы компиляции, сборки, тестирования.

Большинству современных сетевых приложений обычно необходимо осуществлять несколько действий одновременно. В Java реализован механизм поддержки легковесных процессов-потоков (нитей). Многопоточность Java предоставляет средства создания приложений с множеством одновременно активных потоков.

Для эффективной работы с потоками в Java реализован механизм семафоров и средств синхронизации потоков: библиотека языка предоставляет класс Thread, а система выполнения предоставляет средства диспетчеризации и средства, реализующие семафоры. Важно, что работа параллельных потоков с высокоуровневыми системными библиотеками Java не вызовет конфликтов: функции, предоставляемые библиотеками, доступны любым выполняющимся потокам.

По ряду соображений Java более динамичный язык, чем C++. Он был разработан специально для подстройки под изменяющееся окружение. В то время как компилятор Java на этапе компиляции и статических проверок не допускает никаких отклонений, процесс сборки и выполнения сугубо динамический. Классы связываются только тогда, когда в этом есть необходимость. Новые программные модули могут подключаться из любых источников, в том числе, поставляться по сети. В случае с браузером HotJava и другими подобными приложениями интерактивный выполняемый код может быть загружен откуда угодно, что позволяет производить прозрачные модификации приложений. В результате возможно создание интерактивных служб, безболезненно модифицируемых, обслуживающих большое количество клиентов и обеспечивающих развитие электронного бизнеса через Internet.

Если описанные выше характеристики рассматривать по отдельности, то их можно найти во многих программных платформах. Радикальное новшество заключается в способе, предлагаемом Java и системой выполнения, который сочетает в себе все характеристики для предоставления гибкой и мощной системы программирования.

Разработка приложений на Java приводит к получению программного

обеспечения, которое:

- переносимо на разные архитектуры, операционные системы и графические пользовательские интерфейсы
- безопасно
- высокопроизводительно

Благодаря Java работа по разработке программного обеспечения значительно упрощается, все старания направлены на достижение конечной цели: вовремя получить передовой продукт, опирающийся на солидную основу Java.

Язык Java построен с использованием концепций, заимствованных из других языков, таких как C, C++, Eiffel, SmallTalk, Objective C и Cedar/Mes. Поэтому неудивительно, что Java может решать те же задачи, что и эти языки. К примеру, на языке C++ можно создавать утилиты командной строки, библиотеки классов, GUI-приложения и различные другие программы. В этом смысле возможности Java ничем не отличаются от возможностей этих языков. Ниже перечислены четыре типа приложений, которые можно создавать с использованием языка Java:

- Апплеты (мини-приложения)
- GUI-приложения
- Приложения командной строки
- Пакеты (библиотеки)

Апплеты по сути являются мини-приложениями, выполняющимися в среде Java-совместимого браузера, например Netscape 2.x/3.x, Microsoft Explorer 3.x или HotJava.

GUI-приложения - это обычные программы, подобные Windows Notepad, которые не требуют для своей работы присутствия браузера.

Приложения командной строки запускаются из строки системного ОС, подобно команде xcopy в среде MS-DOS или ls в системе UNIX.

Пакеты - это не приложения в "чистом виде", а наборы классов (переносимых байт-кодированных файлов Java), содержащихся в одном пакете (package) (напоминающем библиотеку классов C++). Отсутствует

пользовательский формат для пакетов, подобный тем форматам, которые используются со статическими и динамическими библиотеками в различных операционных системах. Реализация приложения на языке Java намного проще и более мобильна.

Как правило, все классы, относящиеся к некоторому пакету, помещаются в один каталог. Например, все классы, относящиеся к пакету Java Abstract Window Toolkit (AWT - Оконный пользовательский интерфейс), java.awt, расположены в подкаталоге AWT каталога C:\JAVA\CLASSES.

Использованные в разработке компоненты (пакеты).

В данной программе реализовано 10 классов, осуществляющих работу с пользователем и сервером. В написание программы использовались следующие компоненты: java.awt (Abstract Windowing Toolkit), javax.swing, java.net, java.io.

AWT предоставляет интерфейс (API) для стандартных компонент пользовательского интерфейса (кнопки, меню и т.д.). AWT решает одну из основных проблем – предоставление разработчикам платформо-независимого инструмента. AWT содержит как низкоуровневые компоненты (управление клавиатурой, мышкой и т.д., простые графические функции), так и высокоуровневые графические библиотеки.

Java.io обеспечивают работу с потоками ввода/вывода, а также предоставляет стандартные функции работы с файловой системой.

Java.net отвечает за выполнение множества задач, связанных с написанием сетевых приложений. В настоящее время клиент-серверные технологии нашли применение в большинстве корпораций. Главным достоинством этой технологии является то, что процесс обработки данных распределяется между клиентом и сервером. Клиент - это любая программа (GUl-приложение, Telnet и т. д.), запрашивающая обслуживание у серверного приложения. Примерами серверных программ могут служить серверы баз данных, серверы приложений, коммуникационные серверы (FTP, Telnet, Web) и др. Java имеет классы как для клиентских, так и для серверных программ. Java-приложения можно

использовать и как клиенты, и как серверы; апплеты же можно использовать только в качестве клиентских программ. В пакете java.net имеются классы, необходимые для разработки клиент-серверных приложений. В приложении использовался класс Socket – один из фундаментальных блоков, который позволяет устанавливать peer-to-peer соединения.

В ранних - 1.0.x - версиях Java Development Kit активно использовались "тяжелые" компоненты AWT, довольно-таки плотно завязанные с аппаратными платформами. Дальнейшее развертывание концепции "write once, run everywhere" (написать однажды, запускать везде) привело к тому, что в версии 1.1.x наметился переход к таким компонентам, которые бы не были завязаны на конкретные "железо" и операционные системы. Вот здесь и появились те самые "легкие" интерфейсные классы, в которых любой компонент на экране создается средствами Java с помощью графических классов и методов. Такого рода классы компонентов были объединены в библиотеку под названием Swing.

Одна из необходимейших вещей, встроенных в Swing, - система быстрой настройки интерфейса pluggabe Look and Feel (L&F). С ее помощью вы можете изменить внешний вид Java-приложения, написав всего несколько строчек. К примеру, одна и та же программа может выглядеть и как обычное Windows-приложение, и как Unix-приложение с интерфейсом Motif и т. д. Для кроссплатформного использования по умолчанию применяется интерфейс под названием Metal, напоминающий металлический лист с гравировкой.

Описание классов

Client – основной класс, инициализирует окна ввода и вывода. При закрытии одного из них заканчивает выполнение.

Attributes

Name	Class	Type	Initial Value
VERSION	Client	Logical View::java::lang::String	"0.1"
user	Client	Logical View::java::lang::String	new String ("")
PORT	Client	Int	8080

Атрибут VERSION содержит версию программы и служит в информативных целях. User – имя пользователя, работающего в данный момент. PORT –

соответственно номер порта, к которому подключается клиент.

Operations

Name	Signature	Class
Client	Client (Socket s)	Client
checkValidCmdLine	void checkValidCmdLine (Logical View::java::lang::String[] args)	Client
openConnection	Socket openConnection (Logical View::java::lang::String arg)	Client
main	void main (Logical View::java::lang::String[] args)	Client

Main – основная процедура пакета, атрибутом которой является командная строка при запуске. Процедура проверяет корректность командной линии (checkValidCmdLine), устанавливает соединение (openConnection) и инициализирует создание окон. Конструктор Client инициализирует самого клиента (создание и вывод окон).

Класс Cleint связан с классами java.awt.Font (константа displayFont) и ClientInput, ClientOutput (классы, описывающие окна, используемые в работе клиента).

AbstractOutput – класс, используемый для вывода сообщений о работе программы (в debugWindow или stdout).

Attributes

Name	Class	Type	Initial Value
SYS_MSG	AbstractOutput	int	1
USER_MSG	AbstractOutput	int	2
MISC_MSG	AbstractOutput	int	3
NOT_VERBOSE	AbstractOutput	int	0
SYS_VERBOSE	AbstractOutput	int	1
USER_VERBOSE	AbstractOutput	int	2
MISC_VERBOSE	AbstractOutput	int	3
method	AbstractOutput	Logical View::java::lang::String	new String ("stdout")
verboseLevel	AbstractOutput	int	0

Константы *_MSG определяют типа сообщения. (От типа сообщения способ его обработки). Атрибут method – строка содержащая информацию, о том, в какой поток вывода следует направить текстовую информацию (stdout или

textarea). verboseLevel – уровень подробности вывода информации. Также есть атрибут outputField типа javax.swing.JTextArea

Operations

Name	Signature	Class
setVerboseLevel	void setVerboseLevel (int level)	AbstractOutput
setOutput	void setOutput (JTextArea output)	AbstractOutput
setOutput	void setOutput (Logical View::java::lang::String outputMethod)	AbstractOutput
output	void output (Logical View::java::lang::String text, int msgPriority)	AbstractOutput

Операции setVerboseLevel и setOutput изменяют атрибуты с соответствующими именами. А операция output выполняет основную задачу вывода в указанный в method поток вывода информации, связанной с работой программы.

ClientInput – класс ввода сообщений и отправки их серверу.

Operations

Name	Signature	Class
ClientInput	ClientInput (Socket s, JTextField input)	ClientInput
jbInit	void jbInit ()	ClientInput
this_windowClosing	void this_windowClosing (WindowEvent e)	ClientInput
sendButton_mouseClicked	void sendButton_mouseClicked (MouseEvent e)	ClientInput
InputField_keyPressed	void InputField_keyPressed (KeyEvent e)	ClientInput
sendToServer	void sendToServer (Logical View::java::lang::String str)	ClientInput

В работе класса создаются следующие атрибуты socket (java.net.Socket), inputField (javax.swing.JTextField), sendButton (javax.swing.Jbutton).

Операция jbInit отвечает за инициализацию всех графических компонентов. This_windowClosing вызывается при закрытии окна для корректного завершения работы программы. sendButton_mouseClicked – операция, вызываемая при нажатии пользователем на кнопку. InputField_keyPressed обрабатывает символы вводимые с клавиатуры в текстовом поле. sendToServer соответственно отправляет в установленное с сервером соединение текстовую строку введенную пользователем.

ClientOuput – класс, отвечающий за вывод сообщений,

включение/выключение звука, закрытия/открытия debugWindow, сохранения/загрузки лог-файла.

Этот класс имеет большое количество связей, ассоциаций с другими классами, в частности, с Client и overwriteDialog за счет того, что эти классы содержать представители класса ClientOutput. В работе класса используются следующие атрибуты: socket (представитель класса Socket) – сокет, через который проходит общение с сервером, а также объект класса ClientReader – reader (окошко для ввода текста). Также класс связан с другими стандартными классами Java за счет, используемых атрибутов chatLog (javax.swing.JTextArea) – текстовое поле для текста, chatlogScrollPane (javax.swing.JScrollPane), двух объектов класса JPanel: textSetting и settingsPanel и других стандартных объектов swing.

Operations

Name	Signature	Class
ClientOutput	ClientOutput (Socket s, JTextField input)	ClientOutput
jbInit	void jbInit ()	ClientOutput
setPopupFont	void setPopupFont (Font font)	ClientOutput
this_windowClosing	void this_windowClosing (WindowEvent e)	ClientOutput
combo_textsize_itemStateChanged	void combo_textsize_itemStateChanged (ItemEvent e)	ClientOutput
soundToggle_mouseClicked	void soundToggle_mouseClicked (MouseEvent e)	ClientOutput
debugButton_mouseClicked	void debugButton_mouseClicked (MouseEvent e)	ClientOutput
chatLog_mouseClicked	void chatLog_mouseClicked (MouseEvent e)	ClientOutput
popupClear_actionPerformed	void popupClear_actionPerformed (ActionEvent e)	ClientOutput
popupSave_actionPerformed	void popupSave_actionPerformed (ActionEvent e)	ClientOutput
getChatLog	JTextArea getChatLog ()	ClientOutput

Конструктор класс имеет два параметра типа socket – сокет с информацией о соединении и JTextField – используется для смены шрифта в окне. Операция jbInit, соответственно, отвечает за инициализацию всех графических объектов и в случае ошибки генерирует прерывание. Операция setPopupFont выполняет рабочую функцию по смене шрифта, ее параметром является только сам шрифт, передаваемый в качестве параметра. Операция this_windowClosing,

которая вызывается в случае закрытия основного окна, закрывает сокет, производит «сборку мусора» и завершение программы. combo_textsize_itemStateChanged, soundToggle_mouseClicked, debugButton_mouseClicked, chatLog_mouseClicked, popupClear_actionPerformed, popupSave_actionPerformed тоже производят простую работе по выполнению соответствующих действий при некотором определенном событии (нажатии на кнопку или изменению значения в соответствующем меню).

ClientReader – класс чтения потока из сокета, предоставленного сервером.

Класс связан с классом ClientOutput тем, что в этом классе идет инициализация объекта reader рассматриваемого класса. Кроме того класс связан с классами Socket и JTextArea за счет того, что в работе используются атрибуты s и serverMessages соответсвующих типов. К тому же есть булевый атрибут barkOnNewMessage, по значению которого принимается решение подавать ли звуковой сигнал при каждом новом считанном сообщении.

Attributes

Name	Class	Type	Initial Value
barkOnNewMessage	ClientReader	boolean	true

Operations

Name	Signature	Class
ClientReader	ClientReader (Socket socket, JTextArea messages)	ClientReader
working	void working ()	ClientReader
getNotifySettings	boolean getNotifySettings ()	ClientReader
swapNotifySettings	void swapNotifySettings ()	ClientReader
bark	void bark ()	ClientReader
run	void run ()	ClientReader

Операция working выполняет основную задачу: инициализацию потока, чтение новых строк с сервера, вывод их в окно и завершение работы в случае, если дальнейшее чтение сообщений с сервера невозможно (вырабатывает исключение: сервер отключился). Операции getNotifySetting и swapNotifySettings получают и изменяют текущее значение атрибута barkOnNewMessages. В случае, если barkOnNewMessages = true, при

считывание с сервера нового сообщения каждый раз вызывается функция bark, проигрывающее звуковой файл. Run – функция, которая работает до отключения от сервера, из нее вызывается working() и в ее теле проходит обработка исключительных ситуаций.

debugWindow– класс окна отладки.

Атрибуты класса – целочисленное значение открытых окон отладки windowCount, а также debugText и debugInfo – представители JTextArea и JScrollPane. Операции jbInit и this_windowClosing выполняют те же функции, что и в предыдущих рассмотренных классах. addText же просто выводит в окне текст, передаваемый в качестве параметра.

Attributes

Name	Class	Type	Initial Value
windowCount	debugWindow	int	0

Operations

Name	Signature	Class
debugWindow	debugWindow (Logical View::java::lang::String title)	debugWindow
jbInit	void jbInit ()	debugWindow
addText	void addText (Logical View::java::lang::String text)	debugWindow
this_windowClosing	void this_windowClosing (WindowEvent e)	debugWindow

LogFileWriter– класс записи лог-файла.

Этот класс связан с классом overwriteDialog за счет того, что там используется объект writer, представитель класса LogFileWriter. Кроме того этот класс связан с классами File и JFileChooser за счет того атрибутов filename и chooser.

Класс содержит также внутренний класс LogFilter, расширяющий стандартный класс FileFilter для того, чтобы работа производилась с нужными файлами.

Attributes

Name	Class	Type	Initial Value
resultOfDialog	LogFileWriter	int	
extension	LogFileWriter	Logical View::java::lang::String	

Operations

Name	Signature	Class

LogFileWriter	LogFileWriter (Logical View::java::lang::String ext)	LogFileWriter
showSaveDialog	void showSaveDialog (JFrame caller)	LogFileWriter
choosedFile	boolean choosedFile ()	LogFileWriter
askForOverwrite	boolean askForOverwrite (ClientOutput caller)	LogFileWriter
writeLogfile	void writeLogfile (JTextArea messages)	LogFileWriter

overwriteDialog– класс диалога перезаписи файла. Этот класс использует ряд компонентов swing: JLabel, JPanel, JButton и BorderLayout из awt. В булевом атрибуте value хранится значение, определяющая какая кнопка была нажата в диалоге о перезаписи файла (устанавливается в buttonYes_mouserClicked или buttonNo_mouseClicked). Операция jbInit инициализирует все графические объекты, а closeDialog отвечает за корректное завершение диалога.

Attributes

Name	Class	Type	Initial Value
value	overwriteDialog	boolean	false

Operations

Name	Signature	Class
overwriteDialog	overwriteDialog (ClientOutput caller, LogFileWriter writer)	overwriteDialog
jbInit	void jbInit ()	overwriteDialog
buttonYes_mouseClicked	void buttonYes_mouseClicked (MouseEvent e)	overwriteDialog
buttonNo_mouseClicked	void buttonNo_mouseClicked (MouseEvent e)	overwriteDialog
closeDialog	void closeDialog ()	overwriteDialog

userInput– класс окна ввода имени пользователя.

Атрибуты этого класса – строковые значение user и ctrl (имя пользователя и вспомогательная информация, в которой хранится информация о проверке имени). Для того чтобы получить значение переменной user используется операция getUserName, а устанавливается оно операцией OxButton_mouseClicked. Кроме того за счет использования графических объектов класс связан с классами JTextField и JTextButton из библиотеки swing и FlowLayout из библиотеки awt.

Attributes

Name	Class	Type	Initial Value
user	UserInput	Logical View::java::lang::String	

ctrl	UserInput	Logical View::java::lang::String	

Operations

Name	Signature	Class
UserInput	UserInput (Logical View::java::lang::String ctrl)	UserInput
jbInit	void jbInit ()	UserInput
OxButton_mouseClicked	void OxButton_mouseClicked (MouseEvent e)	UserInput
getUsername	Logical View::java::lang::String getUsername ()	UserInput

WindowServant– класс, управляющий закрытием окон.

Единственный атрибут класса – вектор диалоговых окон dialogs. Вектор состоит из объектов типа JFrame. Всего в классе описано 3 операции: addDialog – добавление диалогового окна в вектор, removeDialog – удаление и closeDialogs – закрытие все диалоговых окон, перечисленных в dialogs (используется при завершении работы программы).

Attributes

Name	Class	Type	Initial Value
dialogs	WindowServant	Logical View::java::util::Vector	new Vector ()

Operations

Name	Signature	Class
addDialog	void addDialog (JFrame dialog)	WindowServant
removeDialog	void removeDialog (JFrame dialog)	WindowServant
closeDialogs	void closeDialogs ()	WindowServant

3. Руководство пользователя.

Введение

Данный программный продукт служит для мгновенного обмена сообщений между пользователями. Основными его преимуществами являются минимальный размер, кроссплатформенность, нетребовательность в ресурсах.

Системные требования

• Компьютер с процессором семейства Intel;

• Предустановленный пакет Java Runtime Edition с соответствующей Операционной системой;

• VGA-совместимый монитор;

• 16 мегабайт ОЗУ;

• 1 мегабайт пространства на жестком диске;

• Подключение к сети Интернет или локальной сети

посредством сетевой карты или модема.

Установка и запуск.

Программа не требует никакой предварительной установки. Разархивируйте архивный файл, содержащий файлы скомпилированные для вашей операционной системой.

В новой директории будет находиться скомпилированный Java-пакет с классами chat.jar и в зависимости от вашей операционной системы скрипт для запуска:

Client.bat – для Windows

Client.sh – для Linux или Solaris

Для того, чтобы запустить чат, вам необходимо запустить этот исполняемый скрипт.

Использование клиента.

После запуска Чат-клиента вы увидите небольшое окно, состоящее из одной строки. Оно необходимо для ввода псевдонима, от имени которого будут отправляться сообщения в последующем. Ввод псевдонима является обязательным, так как наличия неадресуемого пользователя в чате недопустимо.

После этого вы увидите основной интерфейс программы – большое многострочное окно с набором управляющих кнопок в нижней части. Соединение с сервером произойдет автоматически. Новые сообщения будут появляться внизу. Помимо этого, окно с поле для ввода Пользователя изменится на окно для ввода сообщений.

Для отправления сообщения наберите его в окне набора и нажмите кнопку Enter. После этого сообщение будет отправлено на сервер и отображено у Вас в окне чата.

Для изменения шрифта основного окна выберите в интерфейсном меню один из шрифтов, а также необходимый размер в выпадающем меню размеров. Изменения произойдут автоматически.

Для включения/отключения звукового оповещения нажмите на кнопку с изображением ноты. Кнопка меняет свой вид на перечеркнутый в отключенном состоянии.

Для включения отключения окна контроля за выполнением программы нажмите на кнопку с изображением жука (bug).

Кнопка с надписью Log откроет отдельное окно с текстовым полем заполненным сообщениями из Основного окна чата. Кнопка Save позволит сохранить весь лог в указанный дальше в диалоговом меню текстовый файл. Кнопка Load позволить открыть для чтения любой текстовый файл.

Замечание.

В файле-скрипте запуска (client.bat или client.sh) можно изменить IP-адрес и порт сервера. Однако будьте внимательны при изменение этих параметров.

Техническая поддержка.

В случае возникновения внештатных ситуаций, пожалуйста, скопируйте сообщение из Debug Window и отправьте их разработчикам по адресу vadimir@dzalbo.com.

4. Техническое задание на разработку клиента сетевого чата.

1. ОБЩИЕ СВЕДЕНИЯ

1.1. Наименование продукта

Simple JavaChat client.

1.2. Краткая характеристика области применения

Данный клиент служит для общения по сети при использовании технологии клиент-сервер без установления межклиентских соединений при помощи специально разработанного Simple JavaChat сервера.

2. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

2.1. Документ, на основании которого ведется разработка

Задание на курсовую работу по дисциплине «Объектно-ориентированное программирование».

2.2. Организация, утвердившая документ

Томский политехнический университет.

3. НАЗНАЧЕНИЕ РАЗРАБОТКИ

Данный программный продукт позволяет пользователям осуществлять общение по сети без необходимости установления прямого соединения компьютеров друг с другом.

4. ТРЕБОВАНИЯ К РАЗРАБОТКЕ

4.1. Требования к функциональным характеристикам

4.1.1. Программа должна корректно работать под любой операционной системой с предустановленным пакетом JRE – Java Runtime Edition (Solaris, Windows, Linux).

4.1.2. Должны быть реализованы стандартные функции работы чата: задание псевдонима, отправка и получение сообщений от других пользователей, запись лог-файла разговоров, а также возможность звукового оповещения.

4.1.3. Программа должна позволять пользователю

менять ряд настроек. В частности, имя пользователя в чате, шрифт и размер шрифта текстового поля, возможность включения/выключения звукового оповещения, а также должна быть реализована возможность сохранения лог-файлов.

4.1.4. Программа должна выводить как обработанную информацию (текст самих сообщения), так и вывод абстрактных сообщений о состоянии системы и проводимых в данный момент операциях (реализовать необходимо в разных окнах с возможностью включения/отключения окна «багтрекинга»).

4.1.5. Программа должна успешно выполняться даже при небольшой скорости установления связи с сервером (работа должна осуществляться по протоколу TCP).

4.2. Требования к надежности

4.2.1. Программа должна отработать без сбоев от начала и до конца всего процесса общения.

4.3. Требования к составу и параметрам технических средств

Для функционирования программы необходим компьютер IBM PC совместимый и следующие технические средства:

• процессор Intel или совместимый;

• объем свободной оперативной памяти 500 Кб;

• стандартный VGA-монитор или совместимый;

• стандартная клавиатура;

• подключение к сети (Internet или Intranet).

4.4.Требования к информационной и программной совместимости

Программа должна успешно функционировать при успешно предустановленном Java Runtime Edition. Программа должна успешно работать под любой из операционных систем, для которой выпускается JRE: Windows, Solaris, Linux.

5. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

Предварительный состав программной документации:

- «Техническое задание».
- «Руководство пользователя».
- «Календарный план».

5. Календарный план на выполнение работ.

№	Наименование этапа	Сроки разработки		Ожидаемый результат
		начало	окончание	
1	Постановка проблемы			Начальное описание функциональных требования.
2	Обследование системы			Детальное описание функциональных возможностей.
3	Составление технического задания			Техническое задание, описывающее требования к назначению и функциям новой системы, показывающих как они удовлетворяют достижению поставленных целей.
4	Системный анализ.			Модель системы в предметной области. Диаграмма использования.
5	Системное проектирование			Архитектура системы. Диаграмма кооперации, последовательности.
6	Программное проектирование			Программная модель. Диаграмма классов.
7	Написание программного кода			Написание программного кода, компилирование, составление готовых к использованию пакетов.
8	Тестирование и отладка.			Проверенный продукт, исправленный и переработанный в случае нахождения ошибок в работе.
9	Системное документирование.			Техническая документация, руководство

				пользователя.
10	Гарантийное обслуживание.			Обучение персонала или переработка исходного кода при заявление.

1. На первом этапе мы оцениваем проблемную ситуацию – необходимость написания программного продукта, позволяющего общаться пользователям в сети (локальной или Internet).
2. Для поставленной проблемы рассматриваем варианты решения. Определяем необходимые цели и описываем функциональные возможности систему, которую планируется создать. Поставленные на этом этапе цели используются для составления технического задания.
3. Составление технического задания для решения поставленной проблемы и достижения поставленных целей. Техническое задание должно описывать: требования к назначению и функциям новой системы, показывающих как они удовлетворяют достижению поставленных целей. Так же должна быть представлена спецификация аппаратных и программных средств, необходимых для работы системы.
4. На этом этапе мы проводим «логическое» моделирование, представляем систему в предметной области, определяем объекты, актанты и функции этих актантов в нашей системе.
5. На этом этапе архитектура системы должна быть завершена, определены методы взаимодействия между объектами данной системы в предметной области. Также должно определено как именно в какой последовательности происходит это взаимодействие.
6. На этапе программного проектирования выбирается язык и методы разработки. Описываются классы, которые будут использоваться для разработки системы в соответствии с заранее построенными диаграммами использования и кооперации.
7. Написание программного кода включает в себя программную реализацию в соответствии с ранее составленным техническим заданием.

8. Тестирование и отладка включает в себя проработку всех функциональных возможностей в различных комбинациях и условиях.
9. Системное документирование включает в себя составления всей сопроводительной документации.

6. Заключение.

В ходе написания программы были использованы элементы объектно-ориентированного программирования. При написании программы были получены навыки необходимые для работы написания при помощи Java сетевых кроссплатформенных оконных приложений, программой для работы с UML Rational Rose 2003. Были созданы соответствующие программе диаграммы UML.

7. Список литературы.

1. Документация Rational Rose 2003 Enterprise Edition.
2. Документация Java(TM) 2 SDK.
3. Интернет.

8. Диаграммы UML.

Задание курсовой работы включает создание диаграмм UML следующих типов: Use case diagram (диаграмма использования), Class diagram (диаграмма классов), Collaboration diagram (диаграмма взаимодействия) и Sequence diagram (диаграмма последовательности). Для создания перечисленных диаграмм было использовано ПО Rational Rose 2003 Enterprise Edition компании IBM.

В распоряжение проектировщика системы Rational Rose предоставляет следующие типы диаграмм, последовательное создание которых позволяет получить полное представление о всей проектируемой системе и об отдельных ее компонентах :

- Use case diagram (диаграммы использования);
- Deployment diagram (диаграммы топологии);
- Statechart diagram (диаграммы состояний);
- Activity diagram (диаграммы активности);
- Interaction diagram (диаграммы взаимодействия);
- Sequence diagram (диаграммы последовательности);
- Collaboration diagram (диаграммы сотрудничества);
- Class diagram (диаграммы классов);
- Component diagram (диаграммы компонент).

Use case diagram (диаграммы вариантов).

Этот вид диаграмм позволяет создать список операций, которые выполняет система. Часто этот вид диаграмм называют диаграммой функций, потому что на основе набора таких диаграмм создается список требований к системе и определяется множество выполняемых системой функций.

Каждая такая диаграмма или, как ее обычно называют, каждый Use case – это описание сценария поведения, которому следуют действующие лица (Actors).

Данный тип диаграмм используется при описании бизнес процессов автоматизируемой предметной области, определении требований к будущей

программной системе. Отражает объекты как системы, так и предметной области и задачи, ими выполняемые.

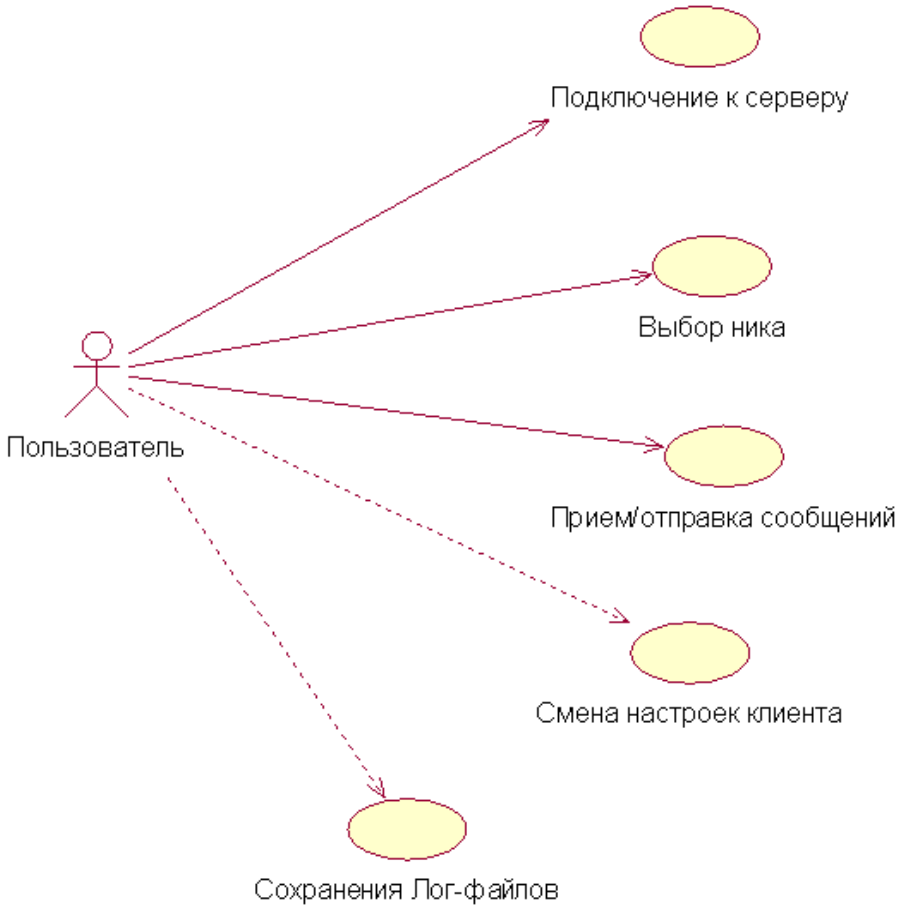

Рис. 1. Диаграмма использования

Единственным актантом нашей системы становится пользователь клиента. Перечислим все варианты использования.

- Подключение к серверу (установление peer-to-peer соединения с сервером, указанным пользователем в качестве атрибута в командной строке при запуске приложения).
- Выбор Ника. Заключается в воде имени в текстовом поле окна, описанного в классе userInput и последующая передача его серверу.

- Прием/отправка сообщений. Это основной режим работы клиента. Пользователь вводит в окне, описанном классом clientInput текст, отправляет его в сокет, предоставленный сервером. В окне, описанном классом clientOutput постоянно выводится текстовая информация полученная с сервера при помощи класса ClientReader.
- Смена настроек клиента. Панель управления в ClientOutput позволяет сменять ряд настроек (включение/выключение звука, отображение/скрытие окна отладки, смена размера шрифта)
- Сохранение лог-файлов. Пользователь в любой момент может сохранить текст из основного окна в текстовый файл или очистить окно от текста.

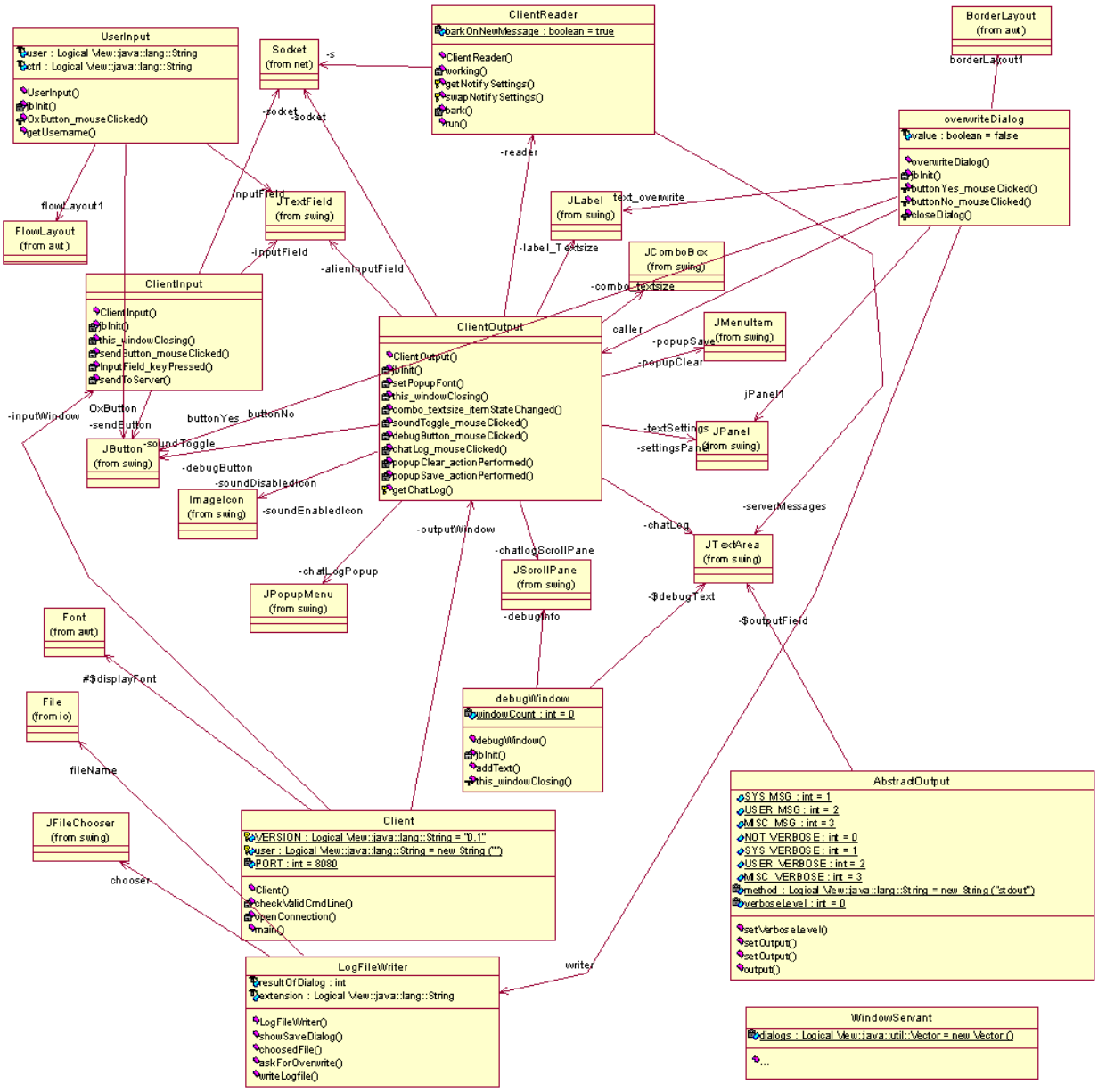

Рис. 2. Диаграмма классов.

Class diagram (диаграммы классов)

Этот тип диаграмм позволяет создавать логическое представление системы, на основе которого создается исходный код описанных классов.

Значки диаграммы позволяют отображать сложную иерархию систем, взаимосвязи классов (Classes) и интерфейсов (Interfaces). Данный тип диаграмм противоположен по содержанию диаграмме Collaboration, на котором отображаются объекты системы. Rational Rose позволяет создавать классы при помощи данного типа диаграмм в различных нотациях. В нотации, предложенной Г. Бучем, которая так и называется Booch, классы изображаются

в виде чего-то нечеткого, похожего на облако. Таким образом Г.Буч пытается показать, что класс – это лишь шаблон, по которому в дальнейшем будет создан конкретный объект.

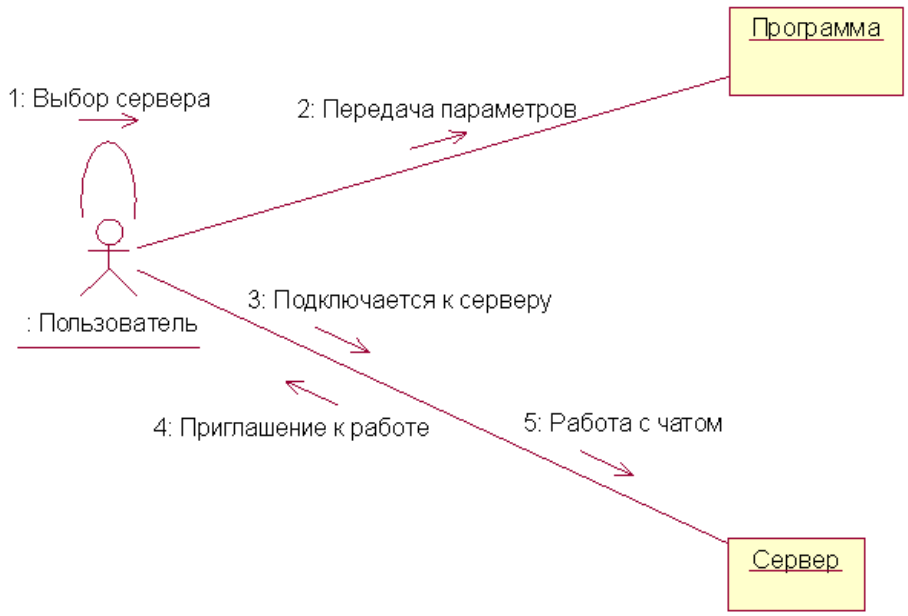

Рис. 3. Диаграмма взаимодействия.

Collaboration diagram (диаграммы взаимодействия).

Этот тип диаграмм позволяет описать взаимодействия объектов, абстрагируясь от последовательности передачи сообщений. На этом типе диаграмм в компактном виде отражаются все принимаемые и передаваемые сообщения конкретного объекта и типы этих сообщений.

По причине того, что диаграммы Sequence и Collaboration являются разными взглядами на одни и те же процессы, Rational Rose позволяет создавать из Sequence диаграммы диаграмму Collaboration и наоборот, а также производит автоматическую синхронизацию этих диаграмм.

Sequence diagram (диаграммы последовательности).

Взаимодействие объектов в системе происходит посредством приема и передачи сообщений объектами-клиентами и обработки этих сообщений

объектами-серверами. При этом в разных ситуациях одни и те же объекты могут выступать и в качестве клиентов, и в качестве серверов.

Данный тип диаграмм позволяет отразить последовательность передачи сообщений между объектами.

Этот тип диаграммы не акцентирует внимание на конкретном взаимодействии, главный акцент уделяется последовательности приема/передачи сообщений. Для того чтобы окинуть взглядом все взаимосвязи объектов, служит Collaboration diagram.

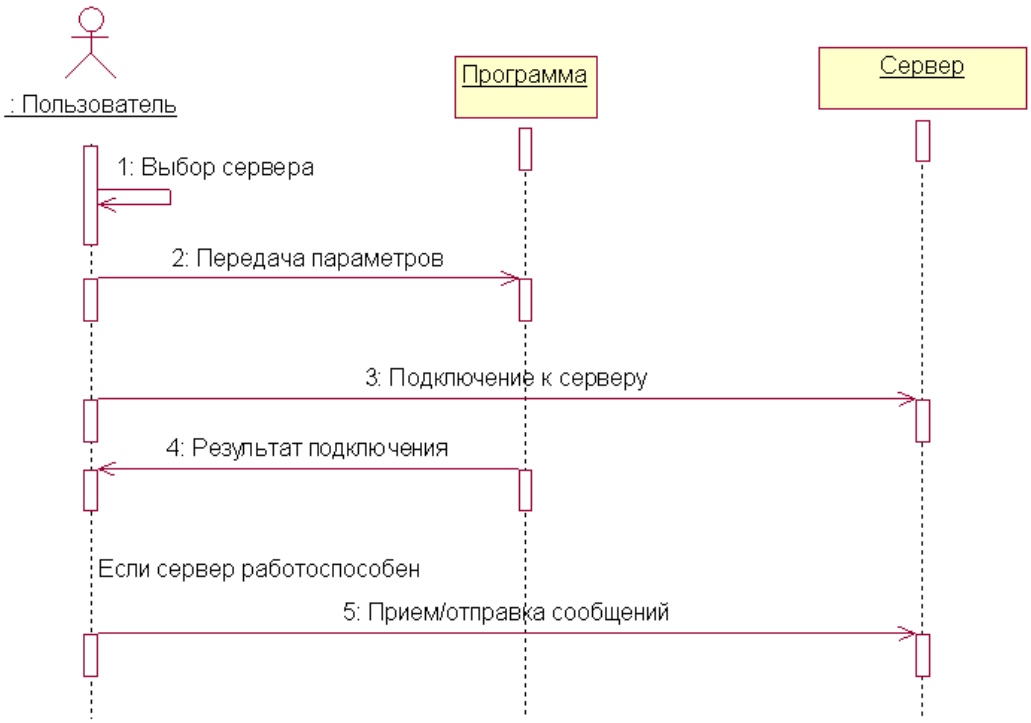

Рис. 4. Диаграмма последовательности.