Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Уральский государственный педагогический университет»

Институт математики, информатики и информационных технологий

Кафедра информатики, информационных технологий
и методики обучения информатике

ОБУЧЕНИЕ БУДУЩИХ ИТ-СПЕЦИАЛИСТОВ РАЗРАБОТКЕ КЛИЕНТ-СЕРВЕРНЫХ ПРИЛОЖЕНИЙ НА ЯЗЫКЕ JAVA

Выпускная квалификационная работа

по направлению «44.03.01 – Педагогическое образование»,
профиль «Информатика»

Исполнитель: студентка группы БИ-41
Глушкова Л.М.

Работа допущена к защите
«____»_____ 2017 г.
Зав. кафедрой _____

Руководитель: к.п.н., доцент кафедры ИИТиМОИ
Газейкина А.И.

Екатеринбург – 2017

Реферат

Глушкова Л.М. ОБУЧЕНИЕ БУДУЩИХ ИТ-СПЕЦИАЛИСТОВ РАЗРАБОТКЕ КЛИЕНТ-СЕРВЕРНЫХ ПРИЛОЖЕНИЙ НА ЯЗЫКЕ JAVA, выпускная квалификационная работа: 54 стр., рис. 3, табл. 1, библ. 40 назв.

Ключевые слова: сервер, клиент, сокеты, потоки, Java, обучение программированию, обучение студентов.

Объект исследования: процесс предметной подготовки будущего ИТ-специалиста.

Цель исследования: разработка лабораторного практикума «Создание клиент-серверных приложения на языке Java» для студентов.

Работа посвящена исследованию углубленного обучения программированию студентов младших и средних курсов с помощью создания клиент-серверных приложений. Обсуждаются проблемы выбора языка программирования и методики обучения. В работе обосновано использование языка программирования Java. Представлен лабораторный практикум «Создание клиент-серверных приложений на языке Java», который состоит из шести лабораторных работ. Студенты, выполняя данные лабораторные работы, узнают основные понятия о сокетах, потоках и архитектуре «клиент-сервер», научатся практически использовать эти знания при создании клиент-серверных приложений на языке программирования Java.

Содержание

Введение

Курс программирования является одним из основных курсов предметной подготовки будущего ИТ-специалиста. В процессе освоения содержания этой дисциплины у студента формируется представление о технологии создания программных средств, без которых невозможны современные информационно-коммуникационные технологии. Курс формирует у студентов представление о способах разработки программных средств современных информационных технологий, учит разрабатывать эти средства на основе использования современных языков, методов и технологий программирования. Поэтому содержание, формы и методы обучения программированию должны соответствовать современному состоянию языков, методов и технологий программирования и перспективам их развития.

Таким образом, актуальность исследования обусловлена необходимостью повышения качества знаний и умений, формирования профессиональных компетенций будущих ИТ-специалистов.

Объект исследования: процесс предметной подготовки будущего ИТ-специалиста.

Предмет исследования: обучение будущего ИТ-специалиста разработке клиент-серверных приложений.

Цель исследования: разработать лабораторный практикум «Создание клиент-серверных приложений на языке Java» для обучения будущих ИТ-специалистов.

В соответствии с поставленной целью определены следующие задачи:

1) Проанализировать содержание, формы и методы обучения студентов программированию.
2) Освоить технологию «клиент-сервер» и ее реализацию на языке Java.
3) Произвести отбор содержания учебного материала для лабораторных работ, разработать их структуру.

4) Разработать лабораторный практикум «Создание клиент-серверных приложений на языке Java» для студентов.
5) Произвести апробацию лабораторного практикума в учебном процессе ИМИиИТ.

Выпускная квалификационная работа состоит из введения, двух глав, заключения, библиографического списка из 40 источников.

Глава 1. Теоретические основы обучения студентов разработке клиент-серверных приложений

1.1. Обучение будущих ИТ-специалистов программированию в вузе

В современном мире информационные технологии развиваются очень стремительно и проникают в большинство сфер жизни общества. В таком случае, чтобы выпускник вуза стал успешным в своей деятельности после обучения, не достаточно освоить лишь базовые знания в области информационных технологий, а также получить навыки поиска готовых решений, а необходимо научиться решать поставленные задачи с помощью программирования. Однако овладение умением программировать остается достаточно сложной задачей для многих студентов. Поэтому содержание, формы и методы обучения программированию как одному из основных предметов подготовки будущих ИТ-специалистов, должны соответствовать современным тенденция развития [5].

Программирование является одним из фундаментальных инструментальных методов современной информатики. Успешность познавательной деятельности студента при обучении программированию зависит от запаса знаний, умения применять их для решения поставленной задачи, опыта ведения самостоятельной познавательной деятельности, интереса к предмету [36].

Однако перед преподавателем, читающим данный курс, встает ряд проблем, решение которых необходимо для эффективной подготовки будущего ИТ-специалиста.

Специфика педагогического вуза такова, что в число первокурсников, как правило, не попадают вчерашние школьники, имеющие высокий уровень знаний и умений в области программирования. Анализ анкет и результатов входных тестов по программированию студентов первого курса Института математики, информатики и информационных технологий Уральского

государственного педагогического университета в течение шести последних лет (это время, когда итоговая аттестация в школе проводилась в форме Единого государственного экзамена, и при зачислении абитуриентов учитывался балл, полученный за ЕГЭ по информатике и ИКТ) позволил заключить, что около 40% студентов первого курса, при обучении в школе, не изучали программирование ни в каком его виде [8].

В качестве входного теста тем первокурсникам, которые занимались программированием в школе, было предложено составить три программы на любом языке программирования (либо алгоритм в виде словесного описания или блок-схемы): осуществить поиск большего из двух чисел; вычислить сумму входящей числовой последовательности; упорядочить элементы массива по возрастанию/убыванию. В результате выяснилось, что десятая часть выпускников помнит только отдельные ключевые слова и команды языка и не в состоянии составить даже простейшей программы (алгоритма); около 50% могут выполнить лишь первое задание на составление разветвляющейся программы; примерно 30% знают, что такое цикл и справились со вторым заданием, и лишь единицы правильно обрабатывают массив и могут реализовать алгоритм его сортировки.

Примерно четвертая часть первокурсников, изучавших в школе программирование, осваивала язык Basic, большая часть обучалась программированию на языке Pascal, отдельные студенты имеют представление о программировании на C/C++. По результатам опроса около 80% первокурсников определили свой уровень подготовки по программированию как нулевой или низкий, остальные – как средний, и никто не оценил как высокий [8].

Таким образом, исходя из перечисленного выше, можно сделать вывод, что в преподавании предмета «информатика и ИКТ» назревают определенные проблемы на уровне обучения в школе, которые позднее проявляются и в обучении в вузе.

В рамках часов, которые отводятся примерной программой базового курса информатики на алгоритмизацию и программирование, является весьма сложным обучить даже основам программирования. Однако ЕГЭ, ГИА, различные олимпиады и конкурсы предусматривают наличие у школьников достаточно уверенного владения приемами программирования [26]. Поэтому, проблему низкой подготовки первокурсников и соответственно более трудоемкого обучения студентов изначально можно решить, работая в системе «школа-вуз», т.е. обучать алгоритмизации и программированию в школе и на более углубленном уровне [24]. Однако немаловажным является и то, на каком уровне владеет программированием учитель в школе, тогда и проблем эффективного использования учебного времени, организации различных кружков и дополнительных курсов в школьном образовании становится значительно меньше. В итоге, если реализовать адаптивную систему обучения «школа-вуз», ориентированную на изучение алгоритмизации и программирования, с подключением квалифицированных преподавателей, увеличением количества учебных часов, то подготовка будущих ИТ-специалистов будет на высоком уровне, но на данный момент проблема остается [28].

Таким образом, уровень подготовки первокурсников в области программирования является крайне низким. Однако вуз также не может дать достаточно глубокие знания в области программирования, ведь часов по изучению данной дисциплины отводится не большое количество.

В процессе обучения в вузе студентам выделяется достаточно большое количество часов на самостоятельную работу. Поэтому, в курсе программирования приоритетное значение получает активная самостоятельная работа студентов по приобретению теоретических знаний и практических умений, осуществляемая под руководством преподавателя. Преподаватель управляет самостоятельной деятельностью студента, оказывая ему консультационно-методическую помощь [34].

Программирование является специфическим предметом, а потому и освоить программирование как вид деятельности становится сложной задачей для студентов. Успешное освоение данной дисциплины зависит не только от приобретенных знаний и умений, а также и от начилия абстрактного стиля мышления. Преподаватель должен ставить перед собой задачу научить не только записи алгоритма на конкретном языке программирования, а показать, что этот язык лишь инструмент и важно осознать, как пользоваться этим инструментом правильно, чтобы студент сам мог конструировать процесс решения поставленной задачи. При правильном подборе теоретического материала и практических заданий у обучающихся сформируется понимание единства принципов функционирования информационных систем [35].

Еще одна проблема в изучении курса программирования – научить размышлять о программировании не только как о техническом прогрессе и решении прикладных задач, а как о виде искусства [17]. Тогда становится важным, в какой форме происходит организация учебной деятельности, а также какова методика преподавания. При традиционной форме обучения, студенты прослушивают теоретический курс и получают блок заданий, но выполнить эти задания в действительности под силу немногим. В результате, большинство студентов выполняют задания механически и не задумываются о закономерностях, которые можно проследить в блоке задач [4]. Притом, что одна из задач преподавателя – это направление познавательной деятельности обучающихся, чтобы каждый самостоятельно задумывался и размышлял над заданием. Именно самостоятельные размышления и поиск решения имеет положительный эффект в процессе обучения, а не заучивание без действительного понимания и механическое нарешивание задач.

Таким образом, можно сделать вывод, что при обучении программированию необходимо развивать следующие умения:

- самостоятельный поиск ответов на вопросы;

- использование всех знаний, а не только тех, что были получены на предыдущем занятии;
- правильное использование инструментов для решения заданий.

В процессе обучения преподаватель является источником знаний, но ни в коем случае он не должен выступать в роли единственного источника. Преподаватель должен показать студентам, что добыть знания они могут и самостоятельно, порекомендовав им различные источники, поставив определенные задачи, чтобы обучающиеся понимали, в каком направлении они должны двигаться. К сожалению, лишь немногие студенты готовы работать в режиме самостоятельной деятельности. Поэтому, преподавателю стоит менять методы обучения, если это требуется [6].

В вузе под методом обучения понимают способы взаимодействия студента и преподавателя, направленные на достижение целей обучения[10].

Можно произвести классификацию методов обучения, с точки зрения использования этих методов в курсе программирования.

По способу передачи информации:

- вербальные;
- наглядные;
- практические.

Вербальные и наглядные методы используются на лекционных занятиях, когда студенты усваивают новый теоретический материал. Основной акцент приходится на практические методы, т.к. при выполнении лабораторных работ или практических заданий студенты приобретают не только новые знания, но и практические навыки. В связи с преобладанием практической деятельности, преподаватель должен консультировать и направлять процесс деятельности обучающихся [22].

По видам дидактических проблем:

- приобретение знаний;
- формирование умений;

- применение знаний;
- контроль знаний, умений и навыков;
- формирование творческой деятельности.

Все перечисленные выше методы применяются, при выполнении студентами лабораторных работ. Преподаватели отмечают, что студенты успешно справляются с заданием по программированию, если материал, необходимый для их решения, рассмотрен на лабораторных или практических занятиях, но испытывают затруднения при выполнении заданий, в которых нужно впервые применить лекционный материал или тем более самостоятельно изучить новый [35].

Часто учебная деятельность представляет собой итерационный поступательный процесс. Рассмотрим итерационный метод обучения.

Метод итераций является универсальным, поэтому его можно применять как на лекционных занятиях, так и при выполнении практических и лабораторных работ по программированию. В лабораторной работе каждое последующее задание является следствием или расширением предыдущего, следовательно, присутствует поступательный итерационный процесс.

Применяя подобную методику обучения, следует отказаться от необоснованно большого количества предлагаемой теории, а с помощью итерационного метода расширять текущие возможности обучающихся, причем введение новых структур и возможностей языка программирования должно обосновываться исходя из необходимости для решений поставленной задачи.

Программирование развивается очень быстро, что требует постоянной актуализации знаний педагогов, а также введения новых форм обучения. На данный момент традиционная методика обучения программированию заключается, прежде всего, в том, что обучаемые знакомятся сначала с теоретическими основами программирования, а затем им предлагается написать программу, используя полученные теоретические знания. Проблема такого подхода заключается в том, что студентам предлагается в большинстве

случаев работа «по образцу», а после сдачи работы они забывают тему и теряют к ней интерес [11].

Программирование является сложной дисциплиной, поэтому важно, чтобы студенты не потеряли мотивацию к ее изучению. Как уже говорилось выше, многие обучающиеся изначально не имеют реального представления о программировании, думая, что через несколько месяцев обучения смогут написать собственную игру и с серьезной проработкой графики, а как только сталкиваются с огромным количеством теории, простыми алгоритмами и программами далекими от своих изначальных представлении, то интерес гаснет, а предмет кажется очень сложным. Поэтому нужно показать начинающему все многообразие программирования в самом начале его изучения, но не скрывая реальных сроков обучения, чтобы достичь такого результата [27].

Можно выделить несколько важных шагов, соблюдая которые преподаватель сможет сохранять и увеличивать мотивацию студентов в обучении программированию:

1) Определить, что именно хотят научиться создавать студенты и дать понять, какие знания и навыки им для этого потребуются.
2) Познакомить с полным теоретическим курсом.
3) Обучать языку программирования как инструменту для решения практических задач.
4) Учить правильно мыслить, проговаривать и обосновывать каждый шаг своего решения. Важно показать, что изначально строится алгоритм решения задачи, а лишь потом возникает вопрос «как реализовать?», а не наоборот.
5) Показать, что полученные знания можно применять не только в рамках учебного процесса, но и в жизни, работе.
6) Максимально сокращать время между идеей решения, реализацией и готовым результатом, хотя бы на первых этапах обучения.

7) Учить писать рациональный код.
8) Научить самостоятельно получать информацию.
9) В рамках курсовых работ и различных проектов помогать в проведении собственного эксперимента, чтобы студент мог написать интересный для него продукт, а не только то, что входит в рамки стандартного курса.

В действительности, если соблюдать каждый шаг, то спустя время можно увидеть в результате мотивированных на обучение студентов. Следовательно, произойдет понимание и осознание значимости предмета, усвоение знаний будет на более высоком уровне, а практические навыки студенты будут совершенствовать не только в рамках учебного процесса. Такой подход гарантирует формирование у студента понимания не вглубь процесса разработки, когда мышление остается на уровне разработки одного конкретного кода, а вширь, смотря на всю индустрию программирования и осознавая практическую значимость своих знаний, наработок. Тогда обучающийся может ставить цель не решить задачу за оценку, а получить знания для достижения практического результата и после окончания вуза.

Как уже говорилось выше, одним из главных условий сохранения интереса к данному предмету у студентов является осознание того, что они могут использовать свои знания на практике в процессе обучения в вузе [13]. Таким образом, целесообразно разработать лабораторный практикум, который позволит студентам получить более глубокие знания по программированию и покажет область их применения на практике.

Разрабатываемый лабораторный практикум должен содержать учебный материал, освоение которого будет способствовать:

- получению новых знаний;
- углублению раннее полученных знаний;
- формированию и развитию умений в области программирования;

- повышению мотивации к изучению программирования за счет возможности продемонстрировать результаты разработки.

Таким образом, нужно произвести отбор материала соответствующего требованиям исследования и разработать лабораторный практикум по программированию для студентов.

1.2. Особенности технологии клиент-сервер и ее реализация на языке программирования Java

В современном мире большинство людей имеют доступ к сети интернет, а значит, каждый подключившийся будет пользоваться технологией «клиент-сервер». Термином «архитектура клиент-сервер» обозначают модель, в которой доступ к базе данных подразделения или к корпоративной базе данных осуществляется с персонального компьютера или рабочей станции [40]. Однако особенных аппаратных средств для архитектуры «клиент-сервер» не требуется. Это программная архитектура, в которой один набор программных компонентов (клиенты) с помощью сообщений обращается к другому набору программных компонентов (серверы) выполнить разноплановые действия. Серверы выполняют затребованные действия и возвращают полученные результаты клиентам. И клиенты, и серверы посылают свои сообщения не по адресам, а по именам. В частности, клиенты посылают свои запросы именованным сервисам, а не конкретным машинам, рассчитывая при этом, что в результате разрешения имен будет определен необходимый физический сервер. Поскольку связь между клиентами и серверами осуществляется с помощью сообщений, клиентская программа и сервис могут работать на разных машинах.

Главной чертой архитектуры «клиент-сервер» является то, что клиент посылает сообщения именованным сервисам. Это дает ряд важных преимуществ [9]:

1) Клиентский и серверный процессы не обязательно должны находиться на одной и той же машине, хотя это вполне допустимо.

2) Клиентский и серверный процессы не обязательно должны работать на однотипных аппаратных средствах или даже под управлением одинаковых операционных систем, если они могут посылать друг другу сообщения. Более того, клиент и сервер не обязательно должны «знать» об аппаратных и программных средах друг друга. Привязка к этим средам – основной источник проблем в приложениях клиент-сервер, потому что какие-либо изменения на клиенте или сервере могут сделать всю систему неработоспособной.
3) Этот процесс может продолжаться до бесконечности: клиент направляет запрос к именованному сервису, который передает его другому именованному сервису, и так далее. Клиент не знает, что первый сервер задействовал других участников.

Для рационального проектирования приложений клиент-сервер должен осуществляться контроль [19]:

- учета числа операций обмена (пар сообщений), особенно там, где может использоваться глобальная сеть;
- отделения задачи управления пользовательским интерфейсом (роли клиента) от задачи управления данными (роли сервера) – нарушение этого принципа является основной причиной высокоинтенсивного графика сообщений между клиентом и сервером.

Проектирование клиент-серверной системы должно происходить таким образом, чтобы и клиент, и сервер выполняли задачи, в которых они сильны, но и в тоже время были освобождены от задач, которые для них сложно реализуемы. Клиент качественно обеспечивает работу прикладного интерфейса, а сервер хорошо выполняет обработку совместно используемых данных. Очевидное решение состоит в следующем – реализовать на клиенте прикладной интерфейс, а на сервере выполнять все операции над совместно используемыми данными [15].

Подробнее рассмотрим, что такое клиент и сервер.

Клиент – это программа, которая запрашивает сервер выполнить то или иное действие (задачу) и вернуть полученные данные клиенту. Клиент подключается к порту, выделенному на сервере [38].

Сервер – это специальная программа, обычно запущенная на отдельном компьютере, которая ожидает от клиентских программ запросы и предоставляет им свои ресурсы в виде данных или сервисных функций. На сервере для работы нужно выделить порт [38].

При выборе системы разработки клиент-серверных приложений важнейшей характеристикой является величина порога, который нужно преодолеть для реализации функций, выходящих за рамки базовых возможностей.

Для этой характеристики существен ряд факторов – открытость всех уровней системы, ограниченный размер компонентов. В тех системах, где есть стандартная оболочка, позволяющая работать с данными, и возможность разрабатывать приложения в рамках оболочки, важно, чтобы вся функциональность этой оболочки была реализована с помощью доступных инструментальных компонентов. Под инструментальностью понимается мера того, в какой степени возможно реализованные нестандартные решения компактно оформить для дальнейшего повторного использования. Это свойство сильно зависит от особенностей используемого разделения системы на компоненты и инструментальности языка разработки [14].

По сочетанию этих параметров наилучшим из средств разработки является язык программирования Java.

К разработкам на базе Visual Basic, Delphi или C++ эти критерии не вполне применимы: для них базовые возможности имеют настолько низкий уровень, что порог начинается почти сразу, и преодолевать его приходится за счет высокой инструментальности C++ как языка программирования, разрабатывая свои библиотеки [14].

Для разработки клиент-серверного приложения на Java нужно использовать многопоточность. В наиболее общем смысле создается поток, реализуя объект класса Thread. В Java для этого определены два способа [30]:

- с помощью реализации интерфейса Runnable;
- с помощью расширения класса Thread.

Самый простой способ создания потока – это объявление класса, реализующего интерфейс Runnable. Интерфейс Runnable абстрагирует единицу исполняемого кода. Можно создать поток из любого объекта, реализующего интерфейс Runnable. Чтобы реализовать этот интерфейс, класс должен объявить единственный метод run( ).

Внутри метода run( ) определяется код, который составляет новый поток. Метод run( ) может вызвать другие методы, использовать другие классы, объявлять переменные – точно так же, как это делает главный поток. Единственным отличием является то, что метод run( ) устанавливает точку входа для другого, параллельного потока внутри программы. Этот поток завершится, когда метод run( ) вернет управление [32].

После того как будет объявлен класс, реализующий интерфейс Runnable, нужно создать объект типа Thread из этого класса. В классе Thread определен конструктор. В этом конструкторе есть экземпляр класса, реализующего интерфейс Runnable. Он определяет, где начнется выполнение потока. После того как новый поток будет создан, он не запускается до тех пор, пока не будет вызван метод start( ), объявленный в классе Thread.

Второй способ создания потока – это объявить класс, расширяющий класс Thread, а затем создать экземпляр этого класса. Расширяющий класс обязан переопределить метод run( ), который является точкой входа для нового потока. Он также должен вызвать метод start( ) для запуска выполнения нового потока [38].

Для того чтобы понять, какой из двух способов создания дочерних потоков лучше, нужно проанализировать собственную программу. Класс

Thread определяет несколько методов, которые могут быть переопределены в производных классах. Из этих методов только один должен быть переопределен в обязательном порядке – это метод run( ). Этот же метод нужен, при реализации интерфейса Runnable. Однако классы следует расширять только в случаях, когда они должны быть усовершенствованы или некоторым образом модифицированы. Поэтому если нет переопределения никаких других методов класса Thread, то лучше просто реализовать интерфейс Runnable. В этом случае, при реализации интерфейса Runnable класс потока не должен наследовать класс Thread, чтобы освободиться от наследования других классов.

Для создания сервера и клиента в Java используются сокеты. Сокеты – это название программного интерфейса для обеспечения обмена данными между процессами. Сокет – абстрактный объект, представляющий конечную точку соединения [14].

Сокеты TCP/IP применяются для реализации надежных двунаправленных, постоянных соединений между точками на основе потоков. Сокет может использоваться для подключения системы ввода-вывода Java к другим программам, которые могут находиться как на локальной машине, так и на любой другой машине в Интернете [32].

В Java существуют два вида сокетов TCP – для серверов и клиентов. Класс ServerSocket является «слушателем», который ожидает подключения клиентов прежде, чем что-либо делать. Класс ServerSocket предназначен для серверов. Класс Socket применяется для клиентов. Он разработан так, чтобы соединяться с серверными сокетами и инициировать обмен по протоколу.

Класс ServerSocket используется для создания для создания серверов, которые прослушивают обращения как локальных, так и удаленных клиентских программ, желающих установить соединения с ними через открытые порты. Класс ServerSocket сильно отличается от обычных классов Socket. Когда создается объект класса ServerSocket, он регистрирует себя в системе в качестве заинтересованного в клиентских соединениях. Конструкторы класса

ServerSocket отражают номер порта, через который будет принимать соединения, а также длину очереди для данного порта. Длина очереди сообщает системе о том, сколько клиентских соединений можно удерживать, прежде чем начать просто отклонять попытки подключения. Если длина очереди не задана, то по умолчанию установлено 50.

Класс ServerSocket включает метод accept(), представляющий собой блокирующий вызов, который будет ожидать от клиента инициации соединений, а затем возвратит обычный объект класса Socket, который далее может служить для взаимодействия с клиентом [38].

Для создания сокетов и управления ими в Java есть специальные классы java.net.Socket и java.net.ServerSocket. Первый для клиента, второй для сервера.

Таким образом, в ходе исследования были изучены и освоены:

- алгоритмы работы сервера и клиента;
- организация многопоточной системы в Java;
- сокеты в Java.

Можно сделать вывод о том, что Java является одним из наилучших средств для разработки клиент-серверных приложений. Используя класс Thread, интерфейс Runnable, а также клиентский и серверный сокеты, можно создать не только обычное клиент-серверное приложение, но и файл-сервер, сервер баз данных, web-сервер и почтовый сервер.

1.3. Лабораторные работы как средство обучения студентов информационным технологиям и программированию

В современном информационном обществе регулярно обновляются средства информационных технологий, что диктует в наши дни новые требования к обучению будущих ИТ-специалистов. Обновленное программное обеспечение, а также обновленные или же совершенно новые языки и среды программирования появляются практически ежегодно. Печатные учебные пособия, которые были актуальны еще несколько лет назад, сейчас уже теряют

свой смысл и не находят применения. В данных условиях нужно искать новые подходы в обучении программированию [37].

Если проанализировать традиционный лекционно-практический метод обучения программированию, то можно выделить ряд существенных недостатков [23]:

- большой процент теоретического материала остается неясным и вызывает затруднения в понимании у студентов, так как отсутствует непосредственная обратная связь с преподавателем;
- теоретический и практический материал не всегда соответствует друг другу
- часто материал, который студент получил на лекции, в итоге или не находит практического применения, или вызывает затруднения в применении;
- нередко теоретический материал, который необходим в решении текущих задач, сообщается на лекциях слишком поздно, в результате чего преподавателю приходится выделять время для теории на практических занятиях;
- отсутствие мотивации к обучению программирования у студентов.

Поэтому нужен новый методический продукт, который решал бы задачу обучения программированию достаточно эффективно и был направлен на взаимодействие студента и преподавателя, чтобы при этом обратной связи студент получал достаточное количество. Таким продуктом может быть система лабораторных работ, которая соответствует следующим требованиям [21]:

- теоретический материал органично сочетается с практической деятельностью;
- задачи сформулированы вначале каждой лабораторной работы, при этом мотивируют студентов к более тщательному изучению теоретического материала;

- вся деятельность в ходе лабораторных работ практикоориентирована;
- в процессе обучения реализована обратная связь между студентом и преподавателем;
- все упражнения в рамках лабораторных работ служат для лучшего осознания теоретического материала, а также закрепления учебных навыков;
- каждая последующая лабораторная работа является логическим продолжением предыдущей;
- теоретический и практический материал выстроен от простого к сложному;
- в самом начале обучения сформирована итоговая цель, студенты, решая текущие задачи, понимают итог своей деятельности.

Средства обучения – такой же важный компонент в образовательном процессе, как и живое общение студентов с преподавателем, а также они являются элементом учебно-материальной базы образовательного учреждения [31].

Лабораторную работу можно рассматривать как средство обучения, как средство формирования самостоятельной активности в поиске информации, как средство формирования понимания значимости предмета, а значит и формирования мотивации к его изучению, как средство формирования необходимых умений и навыков, а также как средство контроля знаний [18]. В ходе выполнения лабораторных работ важным является тот факт, что студенты закрепляют не только практические навыки, но и навыки и умения интеллектуального труда: навыки самостоятельного выполнения учебных заданий, умения рассуждать, наблюдать, экспериментировать, мыслить критически, умения правильно формировать выводы из проделанной работы, умения использовать необходимый инструментарий, умения применять теорию на практике [3].

Для лабораторных работ, как объектов сложных, можно провести классификацию, но важно определить основание, по которому она будет проведена. В качестве основания для классификации лабораторных работ выбраны основные дидактические функции и виды используемых средств.

В соответствии с выбранным основанием классификации лабораторные работы можно разделить на три группы:

1) Лабораторные работы, при выполнении которых требуется применение знаний для решения определенной практической задачи.
2) Лабораторные работы, при выполнении которых требуется подтверждение только что найденной формулы, свойства или многого другого, путем рассмотрения нескольких частных случаев.
3) Лабораторные работы, после выполнения которых, студент должен сам сформулировать вывод о проделанной работе.

Часто преподаватели используют лабораторные работы комбинированного типа, в которых используются элементы нескольких или всех перечисленных выше групп лабораторных работ.

Каждая лабораторная работа содержит в себе определенные этапы самостоятельной или коллективной практической деятельности, а также этап совместной деятельности студентов и преподавателя. Поэтому важно не только грамотно подготовить лабораторные работы, но и провести их. Можно выделить следующие этапы подготовки и проведения лабораторных работ:

- подготовительный этап (преподаватель готовит инструкции, раздаточный материал или другое);
- диалог преподавателя и студентов, рассмотрение всех возникающих вопросов;
- самостоятельное или коллективное выполнение практических заданий;
- консультации преподавателя в процессе выполнения заданий;
- составление и сдача отчета о выполнении лабораторных работ;
- оценка выполнения заданий и последующее обсуждение результатов.

Важной частью каждой лабораторной работы является грамотно сформулированная инструкция к ней. Внутренняя структура лабораторных работ должна быть понятна и логична. Задания, которые входят в лабораторную работу, часто содержат все смысловые опорные пункты, усвоив которые студент понимает всю изучаемую тему в целом. Внутренняя структура лабораторной работы может включать одно или несколько практических заданий, выполнить которые необходимо в аудитории или дома.

Инструкция по выполнению лабораторной работы – это последовательность действий, направленных на достижение цели, которые должны выполнить студенты. Инструкция представляет собой алгоритм, в который входят не только практические задания, но и рекомендации по выполнению каждого из них.

Важно понимать как будет происходить объяснение всех пунктов инструкции. Ведь, студенты должны подходить к выполнению заданий осмысленно, а не механически, иначе формирование умений и навыков не произойдет.

- Объяснение может происходить разными способами:
- Словесное, возможно включение иллюстрации действий;
- Проведение небольшой лекции с дополнительными теоретическими сведениями, которые потребуются при выполнении заданий;
- Разъяснение самых частых ошибок при выполнении заданий;
- Выполнение нескольких заданий совместно с преподавателем, как пример выполнения заданий.

После объяснения заданий студенты приступают к их выполнению. Преподаватель контролирует процесс выполнения лабораторных работ, по необходимости проводит консультации или оказывает индивидуальную помощь.

В итоге можно выделить основные требования к организации лабораторных работ:

1) Лабораторная работа соответствует теме.
2) Лабораторная работа направлена на достижение целей.
3) Инструкция по выполнению лабораторной работы составлена грамотно и понятно для студентов.
4) Перед выполнением лабораторной работы всем студентам предоставляется необходимый инструментарий.
5) Выполнение лабораторных работ ограничено временными рамками.

При выполнении всех требований к организации лабораторных работ, а также правильном их проведении студенты быстро и качественно усвоят материал и смогут самостоятельно сделать выводы о проведенной работе.

Выше были выделены основные этапы проведения лабораторных работ, теперь о некоторых подробнее.

После проведения подготовительных мероприятий педагог переходит ко второму этапу – обсуждение заданий со студентами. На данном этапе является важным выделить цели проведения каждой лабораторной работы. Именно на этом этапе формируется мотивация к изучению данной темы, поэтому преподаватель должен приложить максимум усилий и разъяснить все понятно и грамотно, отвечая на каждый задаваемый вопрос. Желательно, чтобы преподаватель самостоятельно выполнил каждый шаг инструкции, а не просто словесно прокомментировал, если объяснять сущность каждого действия слишком растянуто во времени, то, как минимум, нужно обратить внимание на часто совершаемые ошибки на каждом шаге.

Следующий этап – выполнение лабораторных работ. Как только студенты приступают к выполнению заданий, то сталкиваются с рядом трудностей. Преподаватель должен консультировать как всю группу, если большинство студентов допустили подобные ошибки, так и в индивидуальной форме. Главное, чтобы студенты сохраняли некую самостоятельность в выполнении лабораторных работ, поэтому преподавателю стоит контролировать процесс индивидуального выполнения заданий, исключая списывание.

На этапе оглашения результатов каждый студент должен получить список своих ошибок и рекомендации по их устранению, также преподаватель должен дать исчерпывающий ответ на все поставленные вопросы. Работа над ошибками может быть проведена в аудитории, дома или же проведена лишь на уровне беседы, это на усмотрение преподавателя.

В результате выполнения лабораторных работ студент получает не только информацию, ведь она быстро меняется и усовершенствуется, а умение отсеять главное от ненужного, перевести знания в опыт собственной деятельности, что в итоге способствует формированию информационной компетентности – способности рационально использовать собственные знания, работать с информацией, а также владеть информационными технологиями. Обучающийся приобретает жизненноважные компетенции, что формирует способность личности быстро и грамотно реагировать на текущие запросы времени, и дает возможность ориентироваться в современном обществе. В процессе выполнения конкретным студентом определенного набора заданий, проверяются личностные качества обучаемого, в данном случае просматривается связь личностно ориентированного подхода с компетентностным. Студент проявляет компетентность, т.е. способность установления и реализации связи между знанием и умением в конкретной ситуации [39].

Можно сделать вывод, что лабораторная работа способствует достижению следующих целей:

1) Образовательные. Усвоение знаний и принципов действия, формирование практических умений и навыков использования различного инструментария, совершенствование текущих знаний и обучение их применению на практике.
2) Воспитательные. Формирование ответственности, активизация деятельности исследовательского характера.

3) Развивающие. Развитие интереса к предмету, наблюдательности и умения выдвигать и опровергать гипотезы.

В настоящее время применяется большое количество разнообразных методик обучения, но все их объединяет одно – они направлены на получение человека способного мыслить оригинально, находить нестандартные решения, видеть ассоциации, рассуждать и анализировать [16]. При этом нужно отметить тот факт, что чаще всего для достижения целей, на обучающихся направляют интенсивный поток информации, с которым справиться может лишь малый процент студентов. Такой подход не является эффективным, чтобы решить данную проблему для начала нужно понять сам процесс формирования знаний у человека.

Человека можно рассматривать как сложную динамическую систему. При обучении этой системы, нужно активировать специальную подсистему, которая способна принимать и усваивать информацию. Чтобы управлять процессом усваивания, нужно понять, что представляет собой та информация, которую воспринимает система. Английский ученый М. Полани доказал неоднородность знания, но выделил такой вид знания, который очень сложно передать другим и назвал его неявным, скрытым. В итоге все знания были условно разделены на явные и неявные, которые различны по своей природе и свойствам [25].

С развитием науки, в частности в области искусственного интеллекта, знания разделили на артикулируемые и неартикулируемые. Артикулируемые знания усваиваются без каких-либо затруднений от любого источника. Неартикулируемые знания представляют собой опыт, личностный компонент. Такие знания может добыть только сам студент, в ходе самостоятельного решения практических задач [20].

Создавая методический продукт для обучения важно помнить, что должна присутствовать поддержка неартикулируемой части знаний. Такие задания могут быть в форме небольшого исследования, наблюдения или решения частных случаев с последующим обобщением результатов, главное,

чтобы студент сам мог сформулировать вывод проделанной им работы. Выполняя подобные задания, обучающийся в своем воображении рисует ассоциативную модель, но полнота воспринимаемой информации будет зависеть от ряда факторов. Первый фактор – это изначальная база знаний, если она слишком мала, то и воспринять большой объем информации человек будет не в силах. Второй фактор – как преподаватель преподнес информацию группе, если студент никак не участвовал в диалоге, не увидел цели усвоения знаний или у него не возник интерес, то большая вероятность, что эти знания не будут усвоены. Третий фактор – насколько прочно будут зафиксированы в памяти текущие знания. Третий фактор является самым сложным, чтобы повысить эффективность восприятия нужно, чтобы у студента сформировалась ассоциативная модель в воображении. Соблюдение всех вышеперечисленных факторов способствует повышению качества подготовки будущих специалистов [29].

Благодаря применению системы лабораторных работ в учебном процессе становится возможным переход от иллюстративно-объяснительной функции обучения к инструментально-деятельной и поисковой методике, которая способствует развитию навыков и умений практического использования получаемых знаний [33].

Таким образом, при обучении информационным технологиям и программированию возможно использование лабораторных работ, это поможет в достижении основных дидактических целей, но при этом нужно учитывать, что подготовка, организация и проведение таких работ требует значительных затрат времени и сил.

Глава 2. Разработка лабораторного практикума «Создание клиент-серверных приложений на языке Java»

2.1. Структура лабораторного практикума

Лабораторная работа – вид самостоятельной практической работы, которая проводится для углубления теоретических знаний и развития практических навыков [7].

Лабораторный практикум – несколько лабораторных работ, объединённых по смыслу [7].

Структура лабораторного практикума разработана таким образом, что большая часть теоретического материала дается в самом начале первой лабораторной работы, далее соотношение теоретического и практического материала строится так, что теория дается по необходимости, а в основном упор в сторону практики.

Предлагаем следующую структуру лабораторных работ, так как она соответствует параметрам сохранения мотивации, приобретения новых знаний и умений по принципу от простого к сложному, а также способствует развитию творческих способностей:

1) В начале работы – краткое резюме знаний, умений и навыков, которые приобретут студенты при выполнении этой лабораторной работы.
2) Основная часть работы – выполнение практических заданий для создания приложений. Если студенту нужно перейти в другой класс, то перед текстом вставляется специальное изображение. Если в тексте встречается новый материал, то он выделяется специальной черной рамкой.
3) При выполнении практического задания студентам предлагаются инструкции, прилагаются вставки программного кода для сравнения или подсказок.

4) В третьей, пятой и второй половине шестой лабораторных работ студентам предлагается самостоятельно разработать графический интерфейс приложения, к заданию прилагается пример с комментариями, а так же скриншоты запуска приложений.

Цель лабораторного практикума – сформировать умение разрабатывать клиент-серверные приложения на языке программирования Java.

Содержание материала, который должен быть освоен студентами:

- алгоритм работы сервера;
- алгоритм работы клиента;
- потоки ввода и вывода;
- поток «демон»;
- сокеты;
- реализация многопоточной системы в Java;
- реализация клиентского и серверного сокетов в Java.

Краткое содержание каждой лабораторной работы приведено в таблице 1.

Таблица 1 Лабораторные работы

№	Тема	Краткое содержание
1	Знакомство с технологией «клиент-сервер»	Знакомство с основным теоретическим материалом о технологии «клиент-сервер», потоках и сокетах. Начало разработки первых классов для клиент-серверной системы обмена текстовыми сообщениями.
2	Разработка основных классов для создания клиент-серверной системы обмена текстовыми сообщениями	Знакомство с операторами try и catch, а также созданием потоков ввода и вывода. Завершения разработки основных классов для работы клиент-серверной системы обмена текстовыми сообщениями.
3	Создание графического интерфейса	Создание графического интерфейса для клиент-серверной системы обмена текстовыми сообщениями. Запуск готовой системы.
4	Создание сервера для	Создание основного класса для сервера

	игры «крестики-нолики»	игры «крестики-нолики».
5	Создание графического интерфейса сервера	Создание графического интерфейса для сервера игры «крестики-нолики». Запуск сервера.
6	Создание клиента для игры «крестики-нолики»	Создание основного класса и графического интерфейса для клиента игры «крестики-нолики». Запуск готовой клиент-серверной игры.

Таким образом, разработана структура лабораторного практикума «Создание клиент-серверных приложений на языке Java».

2.2. Лабораторный практикум «Создание клиент-серверных приложений на языке Java»

Введение

При выполнении лабораторных работ студент разработает клиент-серверную систему обмена текстовыми сообщениями и клиент-серверную игру «крестики-нолики».

В каждой лабораторной работе есть теоретический материал, с которым необходимо ознакомиться перед выполнением заданий.

Лабораторная работа №1. Знакомство с технологией «клиент-сервер»

При выполнении этой работы вы:

- познакомитесь с основными понятиями об архитектуре «клиент-сервер»;
- узнаете, что такое потоки, как создается мнопоточность в Java, а также как создать поток «демон»;
- познакомитесь с понятием «сокет» и научитесь его создавать;
- узнаете, какие нужны классы для разработки клиент-серверного приложения, а также подробнее о каждом из них;
- начнете создавать первые классы для клиент-серверного приложения.

Теоретический материал

В архитектуре «клиент-сервер» один набор программных компонентов (клиенты) с помощью сообщений обращается к другому набору программных компонентов (серверы) выполнить разноплановые действия. Серверы

выполняют затребованные действия и возвращают полученные результаты клиентам. И клиенты, и серверы посылают свои сообщения не по адресам, а по именам.

Подробнее рассмотрим, что такое клиент и сервер.

Клиент – программа, которая запрашивает сервер выполнить то или иное действие (задачу) и вернуть полученные данные клиенту. Клиент подключается к порту, выделенному на сервере.

Алгоритм работы клиента выглядит следующим образом:

1) Клиент пытается соединиться с сервером по известному адресу и порту.
2) Клиент создает поток чтения сообщений от сервера.
3) Клиент передает сообщение серверу.
4) Клиент, при получении команды от сервера, выполняет ее.

Сервер – это специальная программа, обычно запущенная на отдельном компьютере, которая ожидает от клиентских программ запросы и предоставляет им свои ресурсы в виде данных или сервисных функций. На сервере для работы нужно выделить порт.

Алгоритм работы сервера выглядит следующим образом:

1) Сервер открывает порт для подключения.
2) Сервер создает поток чтения сообщений от вновь подключенных клиентов.
3) Сервер, при получении команды, выполняет ее и пересылает ответ клиенту.

Для разработки клиент-серверного приложения на Java нужно использовать многопоточность. В наиболее общем смысле создается поток, реализуя объект класса *Thread*. В Java для этого определены два способа:

- с помощью реализации интерфейса *Runnable*;
- с помощью расширения класса *Thread*.

Самый простой способ создания потока – это объявление класса, реализующего интерфейс *Runnable*. Интерфейс *Runnable* абстрагирует единицу исполняемого кода. Можно создать поток из любого объекта, реализующего интерфейс *Runnable*. Чтобы реализовать этот интерфейс, класс должен объявить единственный метод *run( )*.

Внутри метода *run( )* определяется код, который составляет новый поток. Метод *run( )* может вызвать другие методы, использовать другие классы, объявлять переменные – точно так же, как это делает главный поток. Единственным отличием является то, что метод *run( )* устанавливает точку входа для другого, параллельного потока внутри программы. Этот поток завершится, когда метод *run( )* вернет управление.

После того как будет объявлен класс, реализующий интерфейс *Runnable*, нужно создать объект типа *Thread* из этого класса. В классе *Thread* определен конструктор. В этом конструкторе есть экземпляр класса, реализующего интерфейс *Runnable*. Он определяет, где начнется выполнение потока. После того как новый поток будет создан, он не запускается до тех пор, пока не будет вызван метод *start( )*, объявленный в классе *Thread*.

Второй способ создания потока – это объявить класс, расширяющий класс *Thread*, а затем создать экземпляр этого класса. Расширяющий класс обязан переопределить метод *run( )*, который является точкой входа для нового потока. Он также должен вызвать метод *start( )* для запуска выполнения нового потока.

Для создания сервера и клиента в Java используются сокеты. Сокеты – это название программного интерфейса для обеспечения обмена данными между процессами. Сокет – абстрактный объект, представляющий конечную точку соединения.

Сокеты TCP/IP применяются для реализации надежных двунаправленных, постоянных соединений между точками на основе потоков. Сокет может использоваться для подключения системы ввода-вывода Java к

другим программам, которые могут находиться как на локальной машине, так и на любой другой машине в Интернете.

В Java существуют два вида сокетов TCP – для серверов и клиентов. Класс *ServerSocket* является «слушателем», который ожидает подключения клиентов прежде, чем что-либо делать. Класс *ServerSocket* предназначен для серверов. Класс *Socket* применяется для клиентов. Он разработан так, чтобы соединяться с серверными сокетами и инициировать обмен по протоколу.

Класс *ServerSocket* используется для создания для создания серверов, которые прослушивают обращения как локальных, так и удаленных клиентских программ, желающих установить соединения с ними через открытые порты. Класс *ServerSocket* сильно отличается от обычных классов *Socket*. Когда создается объект класса *ServerSocket*, он регистрирует себя в системе в качестве заинтересованного в клиентских соединениях. Конструкторы класса *ServerSocket* отражают номер порта, через который будет принимать соединения, а также длину очереди для данного порта. Длина очереди сообщает системе о том, сколько клиентских соединений можно удерживать, прежде чем начать просто отклонять попытки подключения. Если длина очереди не задана, то по умолчанию установлено 50.

Класс *ServerSocket* включает метод *accept( )*, представляющий собой блокирующий вызов, который будет ожидать от клиента инициации соединений, а затем возвратит обычный объект класса *Socket*, который далее может служить для взаимодействия с клиентом.

Для создания сокетов и управления ими в Java есть специальные классы *java.net.Socket* и *java.net.ServerSocket*. Первый для клиента, второй для сервера.

Создание нужных классов

Для создания сервера нам потребуется три класса.

Задание 1.1.

Запустите Eclipse или IntelliJ IDEA и создайте новый проект, а в нем три класса для работы сервера, как изображено на Рис. 1. Далее нам нужен еще

один проект для работы с клиентом и графическим интерфейсом, создайте классы как изображено на Рис. 2, обратите внимание, что классы разделены на три пакета.

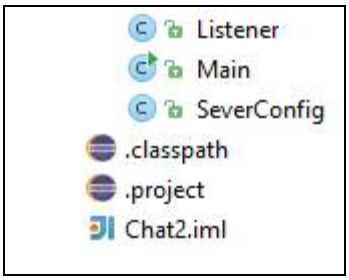

Рис. 1 Классы для создания сервера

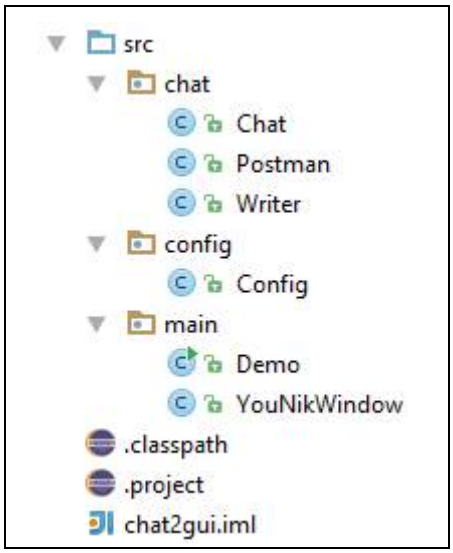

Рис. 2 Классы для клиента и графического интерфейса.

Разработка классов

Разберемся для чего нужен каждый класс:

1. *Listener, Main, SeverConfig* – классы сервера. *Listener* – класс «слушатель», обрабатывает сообщения, полученные от клиентов. *Main* – запускает сервер, ждет подключения новых клиентов. *SeverConfig* – в этом классе будут находится списки клиентов и их имен. Так делать не обязательно, это только для сокращения кода.
2. *Chat, Postman, Writer* – классы клиента. *Chat* – в данном классе создаем клиентский сокет, если не создавать графического интерфейса, то можно в нем сделать метод main и запускать клиента из этого класса. *Postman* – класс «почтальон», передает сообщения классу «слушатель». *Writer* – класс «писатель», считывает сообщение полученное с клавиатуры и передает классу «почтальон».

3. *Config* – класс, в котором будут хранится порт, IP. Так делать не обязательно, но это поможет избежать некоторых ошибок и сократит код.
4. *Demo, YouNikWindow* – классы графического интерфейса и запуска клиента. *Demo* – данный класс запускает клиента, в нем будет описано основное окно чата. *YouNikWindow* – в данном классе будет описано первое окно, в которое клиент будет вводить свой ник.

Задание 1.2

Перейдите в класс *Config*.

Создайте порт и IP-адрес.

```
package config;


public class Config {
    public final int PORT=1234;
    public final String IP="127.0.0.1";

    private static Config ourInstance = new Config();

    public static Config getInstance() {
        return ourInstance;
    }

    private Config() {

    }
}
```

Теоретически порт может быть любым числом от 1 до 65535, но порты от 1 до 1024 используются системными программа и занимать их не стоит, IP 127.0.0.1 используется, если сервер и клиент запускаются с одного компьютера.

Задание 1.3

Перейдите в класс *Chat*.

Создайте клиентский сокет, используя порт и IP из класса *Config*.

После создания сокета может производиться подключение, а также передача и считывание данных, но для этого есть отдельные классы *Postman* и *Writer*, значит нужно создать объекты этих классов.

```
package chat;


import config.Config;

import java.io.IOException;
import java.net.Socket;


public class Chat {
    public Postman postman;
    private Writer writer;

    public Chat() throws IOException, InterruptedException {
        Socket socket = new Socket(Config.getInstance().IP,
Config.getInstance().PORT);
        postman = new Postman(socket);
        writer = new Writer(socket, postman);
        writer.start();
    }

    public void setListener(Writer.WriterListener listener){
        writer.setListener(listener);
    }



}
```

Задание 1.4

Перейдите в класс *Main*.

Создайте серверный сокет, используйте номер порта такой, как и в классе *Config*.

```
public class Main {


   public static void main(String[] args) throws IOException {
      ServerSocket serverSocket=new ServerSocket(1234);
```

Задание 1.5

Перейдите в класс *SeverConfig*.

У сервера есть два списка: клиентские сокеты и имена клиентов. Создайте их.

```
import java.net.Socket;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;


public class SeverConfig {
   public List<Socket> list=new CopyOnWriteArrayList<Socket>();
   public List<String> names=new CopyOnWriteArrayList<String>();
   public boolean run=true;
   private static SeverConfig sign=null;
   private SeverConfig(){}
   public static SeverConfig getSign(){
      if (sign==null){
         sign=new SeverConfig();
      } return sign;
   }

}
```

Задание 1.6

 Перейдите в класс *Main*.

Создайте серверный сокет и метод *accept()*.

Для создания многопоточности нужно объявить класс, расширяющий класс *Thread*, а затем создать экземпляр этого класса. Расширяющий класс обязан переопределить метод *run( )*, который является точкой входа для нового потока. Он также должен вызвать метод *start( )* для запуска выполнения нового потока.

Далее потребуется создать метод *accept()*, представляющий собой блокирующий вызов, который будет ожидать от клиента инициации соединений, а затем возвратит обычный объект класса *Socket*, который далее может служить для взаимодействия с клиентом.

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;


public class Main {


   public static void main(String[] args) throws IOException {
      ServerSocket serverSocket=new ServerSocket(1234);
      String msg="";
      while (SeverConfig.getSign().run) {
         Socket socket;
         socket=serverSocket.accept();
         Listener listener =new Listener(socket);
         SeverConfig.getSign().list.add(socket);
         Thread thread=new Thread(listener);
         thread.setDaemon(true);
         thread.start();
      }
   }

}
```

Вы можете превратить поток в «демон» следующим способом:

```
thread.setDaemon(true);
```

Рис. 3 Демон

Процесс «демон» – это процесс, который выполняет основные сервисные задачи в фоновом режиме, так долго, пока запущена основная программа, но

не является основной частью программы. Таким образом, когда все процессы «не демоны» завершаются, то программа останавливается. И наоборот, пока хоть один процесс «не демон» выполняется, то программа продолжает работу.

Таким образом, в ходе лабораторной работы вы узнали, что архитектура «клиент-сервер» – это один набор программных компонентов (клиенты) с помощью сообщений обращается к другому набору программных компонентов (серверы) выполнить разноплановые действия. Серверы выполняют затребованные действия и возвращают полученные результаты клиентам. И клиенты, и серверы посылают свои сообщения не по адресам, а по именам. Познакомились с классом Thread и интерфейсом Runnable, которые используются для создания многопоточности, а также узнали, что такое поток «демон» и как он работает.

А также узнали, что для разработки клиент-серверного приложения на Java нужно использовать многопоточность. В наиболее общем смысле создается поток, реализуя объект класса *Thread*. В Java для этого определены два способа:

- с помощью реализации интерфейса *Runnable*;
- с помощью расширения класса *Thread*.

Познакомились с понятием сокеты и узнали, как они создаются:

Сокеты – это название программного интерфейса для обеспечения обмена данными между процессами. Сокет – абстрактный объект, представляющий конечную точку соединения.

В Java существуют два вида сокетов TCP – для серверов и клиентов. Класс *ServerSocket* является «слушателем», который ожидает подключения клиентов прежде, чем что-либо делать. Класс *ServerSocket* предназначен для серверов. Класс *Socket* применяется для клиентов. Он разработан так, чтобы соединяться с серверными сокетами и инициировать обмен по протоколу.

Для создания сокетов и управления ими в Java есть специальные классы *java.net.Socket* и *java.net.ServerSocket*. Первый для клиента, второй для сервера.

Разобрались с тем, для чего нужен каждый класс в работе клиент-серверной системы обмена текстовыми сообщениями и начали разработку первых классов.

Лабораторная работа №2. Разработка основных классов для создания клиент-серверной системы обмена текстовыми сообщениями

При выполнении этой работы вы:

- познакомитесь с операторами *try* и *catch*;
- научитесь создавать потоки ввода и вывода;
- завершите разработку основных классов для работы клиент-серверной системы обмена текстовыми сообщениями.

Теоретический материал

Каждый сокет содержит поток ввода и поток вывода класса *InputStream* и *OutputStream*. Любые операции с потоками и сокетами должны выполняться внутри блока try catch.

Операторы try и catch

Для задания блока программного кода, который требуется защитить от исключений, используется ключевое слово *try*. Сразу же после *try*-блока помещается блок *catch*, задающий тип исключения, которое нужно обрабатывать.

Целью большинства правильно сконструированных *catch*-разделов должна быть обработка исключительных ситуаций, чтобы программа продолжила работать и не выводила сообщение об ошибке.

Задание 2.1

Перейдите в класс *Postman*.

Создайте переменные, в которых будут храниться сообщения от клиента. Всего будет 4 вида сообщений.

```
public class Postman {
   public static final String MESSAGE =":msg:";
   public static final String EXIT=":q:";
   public static final String GET_USER=":gu:";
   public static final String HELLO=":hi:";
```

В данном примере *MESSAGE* – это обычное сообщение, *EXIT* – сообщение для выхода из чата (но можно просто закрыть чат и сервер удалит

клиента из списков), *GET_USER* – новый клиент, *HELLO* – оповещение всех клиентов о новом подключении.

Задание 2.2

Перейдите в класс *Writer*.

Writer считывает сообщение клиента с клавиатуры, а *Postman* затем передает его серверу. Значит, нужен не только сокет, но и объект класса *Postman*.

```
package chat;

import java.io.IOException;
import java.io.InputStream;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;


public class Writer extends Thread {
    private Socket socket;
    private Postman postman;
    public boolean isRun = true;
    private WriterListener listener = null;

    public void setListener(WriterListener listener){
        this.listener = listener;
    }

    public Writer(Socket socket, Postman postman) {
        this.socket = socket;
        this.postman = postman;
        this.setDaemon(true);
    }
```

Задание 2.3

Переопределите метод *run()* и создайте поток ввода.

```
@Override
public void run() {
    while (isRun) {
        try {
```

```
            InputStream in = socket.getInputStream();
            Scanner sc = new Scanner(in,"UTF-8");
```

Далее все сообщения «почтальон» должен передать «слушателю», но обратите внимание, что каждый вид сообщений содержит разные системные сообщения, а используя метод *substring()*, можно «вырезать» системное сообщение, чтобы пользователь его не видел.

Напишите обработку всех видов сообщений.

```
while (sc.hasNextLine()) {
                    String str = sc.nextLine();
                    if(listener != null){
                        if (str.contains(Postman.MESSAGE)) {
                            listener.newMessage(str.substring(5)+'\n');
                        } else if (str.contains(Postman.GET_USER)) {
                            String userListStr = str.substring(4).trim();
                            List<String> userList = new
ArrayList<String>();
                            System.out.println(userListStr);
                            while (userListStr.length()>0) {
                                userList.add(userListStr.substring(0,
userListStr.indexOf(",")));
                                userListStr =
userListStr.substring(userListStr.indexOf(",")+1);
                            }
                            listener.postUserList(userList);
                        } else if(str.contains(Postman.EXIT)) {
                            listener.onDisconnect();
                        }
                    }
                    System.out.println(str);
                }
                in.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    public interface WriterListener {
        void newMessage(String msg);
        void postUserList(List<String> userList);
        void onDisconnect();
    }
}
```

Задание 2.4

Перейдите в класс *Config*.

Создайте переменную для хранения имени клиента.

```
public class Config {
    public String myNik = null;
```

Задание 2.5

Перейдите в класс Postman.

Создайте поток вывода.

```
package chat;

import config.Config;

import java.io.IOException;
import java.io.OutputStream;
import java.net.Socket;
import java.nio.charset.Charset;


public class Postman {
   public static final String MESSAGE =":msg:";
   public static final String EXIT=":q:";
   public static final String GET_USER=":gu:";
   public static final String HELLO=":hi:";
   private Socket socket;

   public Postman(Socket socket) {
      this.socket=socket;
   }

    private void send(String msg){
        try {
            OutputStream out = socket.getOutputStream();
            out.write(getUTFByte(msg));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Далее напишите обработку всех видов сообщений.

```
private byte[] getUTFByte(String str){
        return Charset.forName("UTF-
8").decode(Charset.defaultCharset().encode(str)).toString().getBytes();
    }

    public void sendMsg(String str) {
        String msg= MESSAGE +Config.getInstance().myNik+":"+str+'\n';
        send(msg);
    }

    public void sendSysMsg(String str) {
        String msg= MESSAGE +str+'\n';
        send(msg);
    }

    public void helloServer() {
        String msg = HELLO+Config.getInstance().myNik+'\n';
        send(msg);
    }

    public void getUserList(){
        send(GET_USER+"\n");
    }

    public void sendExit(){
        send(EXIT+Config.getInstance().myNik+'\n');
    }

}
```

Задание 2.6

Перейдите в класс *Listener*.

Это класс «слушатель», он принимает сообщения и рассылает их всем клиентам.

В классе *Postman* были созданы переменные для хранения сообщений. Создайте здесь такие же переменные.

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
```

```
import java.util.Scanner;


public class Listener implements Runnable {
    private Socket socket;
    public static final String MESSAGE = ":msg:";
    public static final String EXIT = ":q:";
    public static final String GET_USER = ":gu:";
    public static final String HELLO = ":hi:";
    private boolean run = true;

    public Listener(Socket socket) {
        this.socket = socket;
    }
```

Задание 2.7

Переопределите метод *run()* и создайте поток ввода.

```
@Override
public void run() {
    try {
        String msg;
        InputStream in = socket.getInputStream();
        Scanner scanner = new Scanner(in,"UTF-8");
```

Задание 2.8

Напишите обработку всех входящих сообщений и создайте поток вывода.

Обработка сообщений:

MESSEGE – принять и отправить всем пользователям;

HELLO – принять, внести в список имен, а далее отправить всем пользователям;

GET_USER – принять, это сообщение только для сервера, оно означает, что подключился новый пользователь и его нужно добавить в список сокетов;

EXIT –принять, удалить пользователя из всех списков, прекратить работу с данным клиентом.

```
while (run && SeverConfig.getSign().run) {
                if (scanner.hasNextLine()) {
                    msg = scanner.nextLine();
                    if (msg.contains(MESSAGE)) {
                        msg = msg + '\n';
                        for (Socket soc : SeverConfig.getSign().list) {
                            OutputStream out = soc.getOutputStream();
                            out.write(msg.getBytes("UTF-8"));
```

```
                        }
                    } else if (msg.contains(HELLO)) {
                        String name = msg.substring(4);
                        SeverConfig.getSign().names.add(name);
                        for (Socket soc : SeverConfig.getSign().list) {
                            OutputStream out = soc.getOutputStream();
                            out.write(makeUserList().getBytes());
                        }
                    } else if (msg.contains(GET_USER)) {
                        OutputStream out = socket.getOutputStream();
                        out.write(makeUserList().getBytes());
                    } else if (msg.contains(EXIT)) {
                        run = false;
                        socket.close();
                        int x =
SeverConfig.getSign().list.indexOf(socket);
                        SeverConfig.getSign().list.remove(x);
                        String name = msg.substring(3);
                        x = SeverConfig.getSign().names.indexOf(name);
                        SeverConfig.getSign().names.remove(x);
                        for (Socket soc : SeverConfig.getSign().list) {
                            OutputStream out = soc.getOutputStream();
                            out.write(makeUserList().getBytes());
                        }
                    }
                    System.out.println(msg);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private String makeUserList() throws IOException {
        String str = GET_USER;
        for (String name : SeverConfig.getSign().names) {
            str = str + name + ",";
        }
        System.out.println(str);
        return str+"\n";
    }

}
```

Таким образом, в ходе лабораторной работы вы научились создавать потоки ввода и вывода:

Каждый сокет содержит поток ввода и поток вывода класса *InputStream* и *OutputStream*. Любые операции с потоками и сокетами должны выполняться внутри блока *try catch*.

Познакомились с операторами *try* и *catch*:

Для задания блока программного кода, который требуется защитить от исключений, используется ключевое слово *try*. Сразу же после *try*-блока помещается блок *catch*, задающий тип исключения, которое нужно обрабатывать.

Целью большинства правильно сконструированных *catch*-разделов должна быть обработка исключительных ситуаций, чтобы программа продолжила работать и не выводила сообщение об ошибке.

А также завершили разработку основных классов для клиент-серверной системы обмена текстовыми сообщениями.

Полностью весь практикум находится в приложении.

2.3. Апробация лабораторного практикума «Создание клиент-серверных приложений на языке Java»

Лабораторный практикум «Создание клиент-серверных приложений на языке Java» прошел апробацию студентами второго курса Института математики, информатики и информационных технологий Уральского государственного педагогического университета направлений подготовки педагогическое образование, профиль информатика и информационные системы и технологии.

Целью апробации являлось выявление возможности и целесообразности использования данного лабораторного практикума в учебном процессе вуза.

Студентам было предложено выполнить лабораторный практикум и оценить возможность самостоятельного освоения и доступность изложения материала.

Выполнить полностью лабораторный практикум смогли 17 студентов из 26. Были выявлены несколько недостатков, таких как: темный фон скриншотов, слишком краткий вывод после лабораторных работ. Недостатки были учтены и лабораторный практикум доработан.

Некоторые студенты проявили повышенный интерес к содержанию лабораторных работ и выразили намерение более глубоко освоить реализацию клиент-серверной технологии средствами языка Java.

Таким образом, апробация показала возможность и целесообразность применения разработанного в ходе исследования лабораторного практикума в процессе обучения программированию студентов вуза.

Заключение

В результате выполнения исследования:

- были проанализированы содержание и методы обучения студентов программированию: в результате анализа был сделан вывод о том, что целесообразно разработать лабораторный практикум, который позволит студентам получить более глубокие знания по программированию и покажет область их применения на практике;
- была освоена технология «клиент-сервер» и ее реализация на Java: освоены алгоритмы работы сервера и клиента, организация многопоточной системы в Java, сокеты в Java;
- обоснована целесообразность использования языка программирования Java в рамках обучения программированию будущих ИТ-специалистов: целесообразность использования языка программирования Java обоснована сочетанием таких параметров как открытость всех уровней системы и ограниченный размер компонентов;
- произведен отбор содержания учебного материала для лабораторных работ, разработана их структура: отобран теоретический материал о потоках, сокетах, архитектуре «клиент-сервер», разработана структура рабораторных работ, которая соответствует параметрам сохранения мотивации, приобретения новых знаний и умений по принципу от простого к сложному, а также способствует развитию творческих способностей;
- разработан лабораторный практикум «Создание клиент-серверных приложений на языке Java»: разработан лабораторный практикум из шести лабораторных работ;
- проведена апробация разработанных материалов и проанализированы её результаты: проведена апробация студентами второго курса Института математики, информатики и информационных технологий Уральского

государственного педагогического университета, 17 из 26 студентов выполнили лабораторный практикум полностью, были выявлены и устранены недостатки в практикуме.

Список литературы

1. IntelliJ IDEA: the Java IDE for Professional Developers JetDrains, URL: http://www.jetbrains.com/idea/ (дата обращения: 17.01.2017).
2. Oracle Integrated CloudnApplications and Platform Services, URL: http://www.oracle.com (дата обращения: 17.01.2017).
3. Андрусенко Е. Ю. Формирование учебно-познавательных компетенций учащихся на лабораторных работах по информатике и ИКТ // Историческая и социально-образовательная мысль. Том 7. 2015. 180 – 183 с.
4. Аткинсон Р. Человеческая память и процесс обучения, 1980. 528 с.
5. Барков И. А. Преподавание дисциплины «объектно-ориентированное программирование» // Образовательные технологии и общество. Том 12. 2009. 494 – 500 с.
6. Борисова Е. А. Из опыта обучения программированию на занятиях по информатике в экономическом вузе // Проблемы и перспективы развития образования: материалы междунар. науч. конф. Т. II. — Пермь: Меркурий, 2011. 205 с.
7. Воронько Т. А. Лабораторная работа как средство развития поисковой активности учащихся // Проблемы совершенствования математической подготовки в школе и ВУЗе. М.: Прометей, 2000. 47 с.
8. Газейкина А.И. Обучение программированию будущего учителя информатики // Педагогическое образование в России, 2012. № 5 – Екатеринбург, УрГПУ. 45 – 48 с.
9. Гири Д., Хорстманн К. JavaServer Faces. Третье издание. Издательство: Вильямс, 2011. 544 с.
10. Жужжалов В. Е. Интеграционные методы изучения программирования в вузовском курсе информатики // Вестник МГПУ.

Серия информатика и информатизация образования, 2003. – № 1. – 53 – 54 с.

11. Иванова Л. В. Методика обучения программированию учителей информатики // Ученые записки ЕГПУ. Том 18. Серия «Физико-математические науки» Издательство ЕГПУ, 2010. 138 с.
12. Касьянова Е. В. Методы и средства обучения программированию в ВУЗе // Образовательные ресурсы и технологии, 2016. 25 – 29 с.
13. Ковалев В. И. Мотивы поведения и деятельности, 1992. 192 с.
14. Козополянская Л. М. Использование сокетов для создания клиент-серверного приложения на языке Java // Актуальные проблемы теории и методики обучения информатике, математике и экономике. Том 1. 2016. 37 – 42 с.
15. Козополянская Л. М. Разработка клиент-серверного приложения с использованием сокетов на языке Java // Актуальные вопросы преподавания математики, информатики и информационных технологий. 2016. 45 – 49 с.
16. Лапчик М. П. Методика преподавания информатики. 2009. 217 с.
17. Леви С. Хакеры, герои компьютерной революции, 1984. 16 – 23 с.
18. Малев В. В. Общая методика преподавания информатики: Учебное пособие. - Воронеж: ВГПУ. 2005. 271 с.
19. Мартин К. Соломон, Нирва Мориссо-Леруа, Джули Басу. Oracle. Программирование на языке Java. Издательство: Лори, 2010. 484 с.
20. Микешина Л. А. Философия познания: диалог и синтез подходов // Философия познания. Полемические главы, 2001. №4. 70 – 83 с.
21. Минькович Т. В. Лабораторные и практические работы в методической подготовке учителей информатики // Инновационные проекты и программы в образовании, 2009. №5. 48 – 52 с.

22. Минькович Т. В. Учебно-исследовательская практика в методической подготовке будущего учителя информатики // Муниципальное образование: инновации и эксперимент, 2009. №4. 36 – 39 с.
23. Нежинский В. В., Шумара Е. В. Разработка системы лабораторных работ для обучения программированию // Технические науки, 2016. №7. 112 с.
24. Нельзина О. В. Проблемы обучения программированию по курсу информатики в системе «школа-вуз» // Вопросы Интернет образования, 2006. №13. 27 – 32 с.
25. Полани М. Неявное знание, 1984. 106 с.
26. Родыгин Е. Ф. Методические рекомендации обучения программированию в школе // Вестник Марийского государственного университета, 2011. №7. 20 – 22 с.
27. Рождественская Е. А., Рощина Н. А., Кубарев Е. Н. Особенности мотивации обучения в ВУЗе // Вестник Томского государственного педагогического университета, 2005. №1. 44 – 46 с.
28. Савкина Л. В. Методические проблемы обучения программированию в основной и средней школе, 2013. 15 с.
29. Садилова И. Л., Булатова Е. Г., Черепанов Е. С. Совершенствование обучения в ВУЗе на основе современных подходов // Психология и педагогика: методика и проблемы практического применения, 2010. №13. 237 – 240 с.
30. Сиерра К., Бейтс Б. Изучаем Java. Издательство: Эксмо, 2012 – 720 с.
31. Скаткин М. Н. Совершенствование процесса обучения: учебное пособие// Педагогика, 1971. 121 – 124 с.
32. Снейдер Й. Эффективное программирование TCP/IP. Издательство: Питер, 2001 – 320 с.

33. Стародубцев В. А., Федоров А. Ф. Инновационная роль виртуальных лабораторных работ и компьютерных практикумов // Всерос. науч.-практ. конф. «Единая образовательная информационная среда», Томск, 2003. 79 – 87 с.
34. Титова Г. Ю. О технологии организации самостоятельной работы студентов // Вестник Томского государственного педагогического университета, 2010. №1. 123 – 125 с.
35. Халитова З. Р. Развитие абстрактно-логического мышления будущих учителей информатики при обучении программированию на основе интеграции различных парадигм // Филология и культура, 2012. №1. 273 – 276 с.
36. Халитова З. Р. Развитие познавательной самостоятельности будущих учителей информатики в процессе обучения программированию // Казанский педагогический журнал, 2012. №5. 152 – 158 с.
37. Чермит К. Д., Птущенко Е. Б., Субботина И. П. Инновационный подход в обучении информатике как основа формирования профессиональной информационно-технологической компетентности специалиста // Вестник Адыгейского государственного университета, 2008. №5. 128 с.
38. Шилдт Г. Java. Полное руководство. Восьмое издание. Издательство: Диалектика, 2013 – 1101 с.
39. Щербакова Н. В., Казакова О. Б. Роль практических и лабораторных работ// Вестник Марийского государственного университета, 2011. №7. 16 – 17 с.
40. Эккель Б. Философия Java. Четвертое издание. Издательство: Питер, 2014 – 1168 с.