Федеральное государственное образовательное бюджетное учреждение
высшего образования
«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»
Новороссийский филиал
Кафедра «Информатики, математики и общегуманитарные науки»

И.Г.Рзун

Методические рекомендации

Web-программирование

Направление подготовки: 38.03.05 Бизнес-информатика
Направленность (профиль): ИТ- менеджмент в бизнесе
Форма обучения: очная/заочная/очно-заочная
Квалификация (степень) выпускника: бакалавр

Новороссийск 2019

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ДИСЦИПЛИНЫ СЛУШАТЕЛЯМИ

1. Цели и задачи

Цель дисциплины — Целью курса является изучение средств и методов программирования WEB для овладения знаниями в области технологии программирования; подготовка к осознанному использованию, как языков программирования, так и методов программирования. Одной из целью освоения учебной дисциплины является изучение объектно-ориентированного программирования на языке Java, способов проектирования алгоритмов работы в различной среде: консольной, оконно-графической, многопоточной, сетевой.

При этом основное внимание необходимо уделить не рассмотрению максимально широкого круга вопросов, а на получение студентами глубоких знаний по фундаментальным основам информатики, на формирование у них общего информационного мировоззрения и на развитие алгоритмического мышления.

Задачи дисциплины

Основными задачами курса является:

- изучение объектно-ориентированного программирования на языке Java;
- изучение способов разработки оконно-графического интерфейса программ на языке Java;
- изучение способов создания сетевых программ на языке Java;
- изучение способов создания многопоточных программ на языке Java;
- овладение современными средами разработки программ на языке Java;
- выработка способности к разработке алгоритмических и программных решений в области системного и прикладного

программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям;

- выработка способности критически переосмысливать накопленный опыт, изменять при необходимости вид и характер своей профессиональной деятельности;

- выработка способности к разработке и применению алгоритмических и программных решений в области системного и прикладного программного обеспечения.

Изучение курса позволит студентам получить теоретическую базу, необходимую для успешного усвоения материала учебных дисциплин, связанных с программированием на различных языках программирования в различных средах, а в дальнейшем для их успешной работы и решения производственных задач на ЭВМ.

Студенты должны научиться выполнять разработку программ в различных визуальных средах, разрабатываемых в поддержку современных языков программирования. Уметь пользоваться широким спектром возможностей, предоставляемых этими средами.

2. Место дисциплины в структуре основной образовательной программы.

"Web-программирование" является дисциплиной, углубляющих освоение профиля, образовательной программы высшего образования по направлению подготовки бакалавров 38.03.05 "Бизнес-информатика", профиль «ИТ-менеджмент в бизнесе». Основой для изучения дисциплины являются знания и умения, соответствующие требованиям стандартов основного общего образования по информатике и ИКТ, а также знания и умения, полученные при изучении курсов по программированию и базам данных.

Данный курс рассматриваться в качестве основы для специализированных курсов, ориентированных на более глубокое изучение отдельных направлений Web-технологий.

Знания, полученные при освоении данной дисциплины, необходимы при выполнении выпускной квалификационной работы.

3. Методические указания и порядок изучения дисциплины.

Система обучения основывается на рациональном сочетании нескольких видов учебных занятий (в первую очередь, лекций и практических (лабораторных) занятий), работа на которых обладает определенной спецификой.

Подготовка к лекциям.

Знакомство с дисциплиной происходит уже на первой лекции, где требуется не просто внимание, но и самостоятельное оформление конспекта. Конспектирование лекций – сложный вид аудиторной работы, предполагающий интенсивную умственную деятельность студента. Конспект является полезным тогда, когда записано самое существенное. Не надо стремиться записать дословно всю лекцию. Такое «конспектирование» приносит больше вреда, чем пользы. Целесообразно вначале понять основную мысль, излагаемую лектором, а затем записать ее. Желательно запись осуществлять на одной странице листа или оставляя поля, на которых позднее, при самостоятельной работе с конспектом, можно сделать дополнительные записи, отметить непонятные места.

Конспект лекции лучше подразделять на пункты, соблюдая красную строку. Этому в большой степени будут способствовать вопросы плана лекции, предложенные преподавателям. Следует обращать внимание на акценты, выводы, которые делает лектор, отмечая наиболее важные моменты в лекционном материале замечаниями «важно», «хорошо запомнить» и т.п. Можно делать это и с помощью

разноцветных маркеров или ручек, подчеркивая термины и определения.

Работая над конспектом лекций, Вам всегда необходимо использовать не только учебник, но и ту литературу, которую дополнительно рекомендовал лектор. Именно такая серьезная, кропотливая работа с лекционным материалом позволит глубоко овладеть теоретическим материалом.

Подготовка к практическим (лабораторным) занятиям.

Подготовку к каждому практическому занятию необходимо начать с ознакомления с планом практического занятия, который отражает содержание предложенной темы. Тщательное продумывание и изучение вопросов плана основывается на проработке текущего материала лекции, а затем изучения обязательной и дополнительной литературы, рекомендованной к данной теме. Все новые понятия по изучаемой теме необходимо выучить наизусть и внести в глоссарий, который целесообразно вести с самого начала изучения курса.

Подготовка к лабораторным занятиям и практикумам носит различный характер, как по содержанию, так и по сложности исполнения. Проведение прямых и косвенных измерений предполагает детальное знание измерительных приборов, их возможностей, умение вносить своевременные поправки для получения более точных результатов. Многие лабораторные занятия требуют большой исследовательской работы, изучения дополнительной научной литературы.

В процессе подготовки к практическим занятиям, необходимо обратить особое внимание на самостоятельное изучение рекомендованной литературы. При всей полноте конспектирования лекции в ней невозможно изложить весь материал. Поэтому самостоятельная работа с учебниками, учебными пособиями, научной, справочной литературой, материалами периодических изданий и Интернета является наиболее эффективным методом получения дополнительных

знаний, позволяет значительно активизировать процесс овладения информацией, способствует более глубокому усвоению изучаемого материала.

Защита лабораторных работ должна происходить, как правило, в часы, отведенные на лабораторные занятия. Студент может быть допущен к следующей лабораторной работе только в том случае, если у него не защищено не более двух предыдущих работ.

Рекомендации по работе с литературой.

Работу с литературой целесообразно начать с изучения общих работ по теме, а также учебников и учебных пособий. Далее рекомендуется перейти к анализу монографий и статей, рассматривающих отдельные аспекты проблем, изучаемых в рамках курса, а также официальных материалов и неопубликованных документов (научно-исследовательские работы, диссертации), в которых могут содержаться основные вопросы изучаемой проблемы.

Работу с источниками надо начинать с ознакомительного чтения, т.е. просмотреть текст, выделяя его структурные единицы. При ознакомительном чтении закладками отмечаются те страницы, которые требуют более внимательного изучения.

В зависимости от результатов ознакомительного чтения выбирается дальнейший способ работы с источником. Если для разрешения поставленной задачи требуется изучение некоторых фрагментов текста, то используется метод выборочного чтения. Если в книге нет подробного оглавления, следует обратить внимание ученика на предметные и именные указатели.

Избранные фрагменты или весь текст (если он целиком имеет отношение к теме) требуют вдумчивого, неторопливого чтения с «мысленной проработкой» материала. Такое чтение предполагает выделение: 1) главного в тексте; 2) основных аргументов; 3) выводов. Особое внимание следует обратить на то, вытекает тезис из аргументов или нет.

Необходимо также проанализировать, какие из утверждений автора носят проблематичный, гипотетический характер, и уловить скрытые вопросы.

Понятно, что умение таким образом работать с текстом приходит далеко не сразу. Наилучший способ научиться выделять главное в тексте, улавливать проблематичный характер утверждений, давать оценку авторской позиции – это сравнительное чтение, в ходе которого Вы знакомитесь с различными мнениями по одному и тому же вопросу, сравниваете весомость и доказательность аргументов сторон и делаете вывод о наибольшей убедительности той или иной позиции.

Если в литературе встречаются разные точки зрения по тому или иному вопросу из-за сложности прошедших событий и правовых явлений, нельзя их отвергать, не разобравшись. При наличии расхождений между авторами необходимо найти рациональное зерно у каждого из них, что позволит глубже усвоить предмет изучения и более критично оценивать изучаемые вопросы. Знакомясь с особыми позициями авторов, нужно определять их схожие суждения, аргументы, выводы, а затем сравнивать их между собой и применять из них ту, которая более убедительна.

Следующим этапом работы с литературными источниками является создание конспектов, фиксирующих основные тезисы и аргументы.

Таким образом, при работе с источниками и литературой важно уметь:

– сопоставлять, сравнивать, классифицировать, группировать, систематизировать информацию в соответствии с определенной учебной задачей;
– обобщать полученную информацию, оценивать прослушанное и прочитанное;

- фиксировать основное содержание сообщений; формулировать, устно и письменно, основную идею сообщения; составлять план, формулировать тезисы;
- готовить и презентовать развернутые сообщения типа доклада;
- работать в разных режимах (индивидуально, в паре, в группе), взаимодействуя друг с другом;
- пользоваться реферативными и справочными материалами;
- контролировать свои действия и действия своих товарищей, объективно оценивать свои действия;
- обращаться за помощью, дополнительными разъяснениями к преподавателю, другим студентам;
- пользоваться лингвистической или контекстуальной догадкой, словарями различного характера, различного рода подсказками, опорами в тексте (ключевые слова, структура текста, предваряющая информация и др.);
- использовать при говорении и письме перифраз, синонимичные средства, слова-описания общих понятий, разъяснения, примеры, толкования, «словотворчество»;
- повторять или перефразировать реплику собеседника в подтверждении понимания его высказывания или вопроса;
- обратиться за помощью к собеседнику (уточнить вопрос, переспросить и др.);
- использовать мимику, жесты (вообще и в тех случаях, когда языковых средств не хватает для выражения тех или иных коммуникативных намерений).

Подготовка к промежуточной аттестации.

При подготовке к промежуточной аттестации целесообразно:

- внимательно изучить перечень вопросов и определить, в каких источниках находятся сведения, необходимые для ответа на них;
- внимательно прочитать рекомендованную литературу;

– составить краткие конспекты ответов (планы ответов).

Лекционные занятия проводятся в соответствии с тематическим ланом, при изложении материала рекомендуется использовать презентации в среде PowerPoint и фрагменты печатных материалов по теме лекции.

В ходе интерактивных занятий следует проводить разбор конкретных примеров, максимально приближенных к реальным данным, соответствующих экономической и финансовой информации.

Основное внимание при проведении практических занятий следует уделять развитию навыков формирования рациональных схем данных предметной области, реализации этих схем в среде современных аналитических систем, формирования сложных содержательных запросов по выбору данных, использования методов и алгоритмов анализа данных.

При этом задача состоит в обучении профессиональным навыкам разработки и использования современных аналитических систем.

Поскольку большая часть учебного времени отводится на самостоятельное изучение дисциплины, рекомендуется уделить особое внимание организации и планированию самостоятельной работы, раскрыв существующие возможности созданных в университете корпоративных образовательных ресурсов (электронная библиотека, компьютерные обучающие программы, электронные учебные ресурсы, учебно-методические комплексы (УМК), облачные сервисы).

Практические занятия в компьютерных классах позволяют студентам сформировать навыки работы с современными аналитическими системами на их базе и CASE – системами для решения прикладных экономических задач.

Методика проведения занятий заключается в совместном решении студентами учебной группы под руководством преподавателя типовых задач по изучаемым темам дисциплины, которые далее выполняются на вариантах индивидуальных

данных. Итогом таких занятий является самостоятельное решение студентами задачи на реальных данных.

Внедрение активных и интерактивных элементов в проведение занятий по дисциплине может осуществляться разными методами: семинар с групповым обсуждением, опрос, компьютерный эксперимент и др**.**

Интерактивная форма проведения занятий способствует формированию профессиональных компетенций для успешного освоения основных дисциплин блока программы. Реализация интерактивной формы обеспечивается базой данных прикладной предметной области, коллективной работой над решениями задач, отсутствием единственного решения, единой целью в поиске решения. Конечная цель - выработать у студентов умение реализовывать и оценивать альтернативные варианты различных аспектов функционирования современных аналитических систем.

Тема 1 Введение. Основы web-технологий

Основы языка Java

Содержание темы: Обзор примитивных типов. Основные управляющие конструкции. Структура классов. Наследование и предопределенные классы. Интерфейсы, оболочки. Пакеты.

Методические рекомендации

Программирование – это написание исходного кода программы на одном из языков программирования. Существует множество различных языков программирования, благодаря которым создаются всевозможные программы, решающие определенный круг задач. Язык программирования – это набор зарезервированных слов, с помощью которых пишется исходный код программы. Компьютерные системы не в силах (пока) понимать человеческий язык и уж тем более, человеческую логику (особенно женскую), поэтому все программы пишутся на языках программирования, которые впоследствии переводятся на язык компьютера или в машинный код. Системы, переводящие исходный код программы в машинный код, очень сложные и их, как правило, создают не один десяток месяцев и не один десяток программистов. Такие системы называются интегрированными средами программирования приложений или инструментальными средствами.

Система программирования представляет собой огромную продуманную визуальную среду, где можно писать исходный код программы, переводить его в машинный код, тестировать, отлаживать и многое другое. Дополнительно существуют программы, которые позволяют производить вышеперечисленные действия при помощи командной строки.

Вы, наверное, не раз слышали термин «программа написана под Windows или под Linux, Unix». Дело в том, что среды программирования при переводе языка программирования в машинный код могут быть двух видов – это компиляторы и интерпретаторы. Компиляция или интерпретация программы

задает способ дальнейшего выполнения программы на устройстве. Программы написанные на языке Java всегда работают на основе интерпретации, тогда как программы написанные на C/C++ – компиляции. В чем разница этих двух способов?

Компилятор после написания исходного кода в момент компиляции читает сразу весь исходный код программы и переводит в машинный код. После чего программа существует, как одно целое и может выполняться только в той операционной системе, в которой она была написана. Поэтому программы, написанные под Windows, не могут функционировать в среде Linux и наоборот. Интерпретатор осуществляет пошаговое или построчное выполнение программы каждый раз, когда она выполняется. Во время интерпретации создается не выполняемый код, а виртуальный, который впоследствии выполняется виртуальной Java машиной. Поэтому на любой платформе – Windows или Linux, Java-программы могут одинаково выполняться при наличии в системе виртуальной Java машины, которая еще носит название Системы времени выполнения.

Объектно-ориентированное программирование

Объектно-ориентированное программирование строится на базе объектов, что в кой-то мере аналогично с нашим миром. Если оглянуться вокруг себя, то обязательно можно найти то, что поможет более ярко разобраться в модели такого программирования. Например, я сейчас сижу за столом и печатаю эту главу на компьютере, который состоит из монитора, системного блока, клавиатуры, мыши, колонок и так далее. Все эти части являются объектами, из которых состоит компьютер. Зная это, очень легко сформулировать какую-то обобщенную модель работы всего компьютера. Если не разбираться в тонкостях программных и аппаратных свойств компьютера, то можно сказать, что объект Системный блок производит определенные действия, которые показывает объект Монитор. В свою очередь объект Клавиатура может корректировать или

вовсе задавать действия для объекта Системный блок, которые влияют на работу объекта Монитор. Представленный процесс очень хорошо характеризует всю систему объектно-ориентированного программирования.

Представьте себе некий мощный программный продукт, содержащий сотни тысяч строк кода. Вся программа выполняется построчно, строка за строкой и в принципе каждая из последующих строк кода обязательно будет связана с предыдущей строкой кода. Если не использовать объектно-ориентированное программирование, и когда потребуется изменить этот программный код, скажем при необходимости улучшения каких-то элементов, то придется произвести большое количество работы со всем исходным кодом этой программы.

В объектно-ориентированном программировании все куда проще, вернемся к примеру компьютерной системы. Допустим, вас уже не устраивает семнадцати дюймовый монитор. Вы можете спокойно его обменять на двадцати дюймовый монитор, конечно же, при наличии определенных материальных средств. Сам же процесс обмена не повлечет за собой огромных проблем, разве что драйвер придется сменить, да вытереть пыль из-под старого монитора и все. Примерно на таком принципе работы и строится объектно-ориентированное программирование, где определенная часть кода может представлять класс однородных объектов, которые можно легко модернизировать или заменять.

Объектно-ориентированное программирование очень легко и ясно отражает суть решаемой проблемы и что самое главное, дает возможность без ущерба для всей программы убирать ненужные объекты заменяя эти объекты на более новые. Соответственно общая читабельность исходного кода всей программы становится намного проще. Существенно и то, что один и тот же код можно использовать в абсолютно разных программах.

Классы

Стержнем всех программ Java являются классы, на которых

основывается объектно-ориентированное программирование. Вы по сути уже знаете, что такое классы, но пока об этом не догадываетесь. В предыдущем разделе мы говорили об объектах, ставя в пример устройство всего компьютера. Каждый объект, из которых собран компьютер, является представителем своего класса. Например, класс Мониторов объединяет все мониторы вне зависимости от их типов, размеров и возможностей, а один какой-то конкретный монитор, стоящий на вашем столе и есть объект класса мониторов.

Такой подход позволяет очень легко моделировать всевозможные процессы в программировании, облегчая решение поставленных задач. Например, имеется четыре объекта четырех разных классов: монитор, системный блок, клавиатура и колонки. Чтобы воспроизвести звуковой файл необходимо при помощи клавиатуры дать команду системному блоку, само же действие по даче команды вы будете наблюдать визуально на мониторе и, в итоге, колонки воспроизведут звуковой файл. То есть любой объект является частью определенного класса и содержит в себе все имеющиеся у этого класса средства и возможности. Объектов одного класса может быть столько, сколько это необходимо для решения поставленной задачи.

Методы

Когда приводился пример воспроизведения звукового файла, то было упомянуто о даче команды или сообщения, на основе которого и выполнялись определенные действия. Задача по выполнению действий решается с помощью методов, которые имеет каждый объект. Методы – это набор команд, с помощью которых можно производить те или иные действия с объектом.

Каждый объект имеет свое назначение и призван решать определенный круг задач с помощью методов. Какой толк был бы, например, в объекте Клавиатура, если нельзя было нажимать на клавиши, получая при этом возможность отдавать команды? Объект Клавиатура имеет некое количество клавиш, с помощью которых пользователь приобретает контроль над устройством

ввода и может отдавать необходимые команды. Обработка таких команд, происходит с помощью методов.

Например, вы нажимаете клавишу Esc для отмены каких-либо действий и тем самым даете команду методу, закрепленному за этой клавишей который на программном уровне решает эту задачу. Сразу же возникает вопрос о количестве методов объекта Клавиатура, но здесь может быть различная реализация – как от определения методов для каждой из клавиш (что, вообще-то, неразумно), так и до создания одного метода, который будет следить за общим состоянием клавиатуры. То есть, этот метод следит за тем, была ли нажата клавиша, а потом в зависимости от того какая из клавиш задействована, решает, что ему делать.

Итак, мы видим, что каждый из объектов может иметь в своем распоряжении набор методов для решения различных задач. А поскольку каждый объект является объектом определенного класса, то получается, что класс содержит набор методов, которыми и пользуются различные объекты одного класса. В языке Java все созданные вами методы должны принадлежать или являться частью какого-то конкретного класса.

Синтаксис и семантика языка Java

Для того чтобы говорить и читать на любом иностранном языке, необходимо изучить алфавит и грамматику этого языка. Подобное условие наблюдается и при изучении языков программирования, с той лишь разницей, как мне кажется, что этот процесс несколько легче. Но прежде чем начинать писать исходный код программы, необходимо сначала решить поставленную перед вами задачу в любом удобном для себя виде.

Давайте создадим некий класс отвечающий, например, за телефон, который будет иметь всего два метода: включающий и выключающий этот самый телефон. Поскольку мы сейчас не

знаем синтаксис языка Java, то напишем класс Телефон на абстрактном языке.

```
Класс Телефон
{
Метод Включить()
{
// операции по включению телефона
}
Метод Выключить()
{
// операции по выключению телефона
}
}
```

Примерно так может выглядеть класс Телефон. Заметьте, что фигурные скобки обозначают соответственно начало и конец тела класса, метода, либо всякой последовательности данных. То есть скобки указывают на принадлежность к методу или классу. На каждую открывающую скобку обязательно должна быть закрывающая скобка. Чтобы не запутаться их обычно ставят на одном уровне в коде.
А теперь давайте запишем тот же самый класс только уже на языке Java.

```
class Telefon
{
void on()
{
// тело метода on()
}
void off()
{
// тело метода off()
}
}
```

Ключевое слово class в языке Java объявляет класс, далее идет название самого класса. В нашем случае это Telefon. Сразу пару слов касательно регистра записи. Почти во всех языках программирования важно сохранять запись названий в том регистре, в котором она была сделана. Если вы написали Telefon, то уже такое написание как telefon или TELefoN выдаст ошибку при компиляции. Как написали первоначально, так и надо писать дальше.

Зарезервированные или ключевые слова записываются в своем определенном регистре, и вы не можете их использовать, давая их названия методам, классам, объектам и так далее. Пробелы между словами не имеют значения, поскольку компилятор их просто игнорирует, но для читабельности кода они важны.

В теле класса Telefon имеются два метода: on() – включающий телефон и метод off() – выключающий телефон. Оба метода имеют свои тела и в них по идее должен быть какой-то исходный код, описывающий необходимые действия обоих методов. Для нас сейчас неважно, как происходит реализация этих методов, главное – это синтаксис языка Java.

Оба метода имеют круглые скобки on(), внутри которых могут быть записаны параметры, например on(int time) или on(int time, int time1). С помощью параметров происходит своего рода связь методов с внешним миром. Говорят, что метод on(int time) принимает параметр time. Для чего это нужно? Например, вы хотите, чтобы телефон включился в определенное время. Тогда целочисленное значение в параметре time будет передано в тело метода и на основе полученных данных произойдет включение телефона. Если скобки пусты, то метод не принимает никаких параметров.

Комментарии

В классе Telefon в телах обоих методов имеется запись после двух слэшей: //. Такая запись обозначает комментарии, которые будут игнорироваться компилятором, но нужны для читабельности кода. Чем больше информации вы

закомментируете по ходу написания программы, тем больше у вас будет шансов вспомнить через год, над чем же все это время трудились.

Комментарии в Java могут быть трех видов, это:

//, /*…*/ и /**…*/

Комментарии, записанные с помощь оператора // должны располагаться в одной строке:

```
// Одна строка
!!! Ошибка! На вторую строку переносить нельзя!
// Первая строка
// Вторая строка
// …
// Последняя строка
```

Комментарии, использующие операторы /*…*/ могут располагаться на нескольких строках. В начале вашего комментария поставьте /*, а в конце, когда закончите комментировать код, поставьте оператор */. Последний вид комментария /**…*/ используется при документировании кода и также может располагаться на любом количестве строк.

Типы данных Java

Чтобы задать произвольное значение, в Java существуют типы данных. В классе Telefon мы создали два метода. Оба метода не имели параметров, но когда приводился пример метода on(int time) с параметром time, говорилось о передаче значения в метод. Данное значение указывало на время, с помощью которого якобы должен включиться телефон. Спецификатор int как раз и определяет тип значения time. В Java 2 ME шесть типов данных.

• byte – маленькое целочисленное значение от –128 до 128;
• short – короткое целое значение в диапазоне от –32768 до 32767;
• int – содержит любое целочисленное значение от –2147483648 до 2147483647;
• long – очень большое целочисленное значение, от –

922337203685475808 до 9223372036854775807;
• char – это символьная константа в формате Unicode. Диапазон данного формата от 0 до 65536, что равно 256 символам. Любой символ этого типа должен записываться в одинарных кавычках, например: ‘G’;
• boolean – логический тип, имеет всего два значения: false – ложь и true – истина. Этот тип часто используется в циклах о которых чуть позже. Смысл очень прост – если у вас в кармане есть деньги, предположительно это true, а если нет то false. Таким образом, если деньги имеются – идем в магазин за хлебом или пивом (нужное подчеркнуть), если нет денег – остаемся дома. То есть это такая логическая величина, которая способствует выбору дальнейших действий вашей программы.

Чтобы объявить какое-то необходимое значение используется запись:

```
int time;
long BigTime;
char word;
```

Оператор точка с запятой необходим после записей и ставится в конце строки. Можно совместить несколько одинаковых по типу объявлений через запятую:

```
mt time, time1, time2;
```

Теперь давайте, усовершенствуем наш класс Telefon, добавив в него несколько значений. Методы on() и off() нам больше не нужны, добавим новые методы, которые действительно могут решать определенные задачи.

```
class Telefon
{
//S – площадь дисплея
//w – ширина дисплея
//h – высота дисплея
int w, h, S;
//метод, вычисляющий площадь дисплея
```

```
vord Area()
{
S = w*h;
}
}
```

Итак, мы имеем три переменные S, w и h, отвечающие, соответственно, за площадь, ширину и высоту дисплея в пикселях. Метод Area() вычисляет площадь экрана телефона в пикселях. Операция бесполезная, но очень показательная и простая в понимании. Тело метода Area() обрело себя и имеет вид S = w*h. В этом методе мы просто перемножаем ширину на высоту и присваиваем или как еще говорят, сохраняем результат в переменной S. Эта переменная будет содержать значения площади дисплея данного телефона.

Сейчас мы подошли вплотную к операторам языка Java, с помощью которых можно совершать всевозможные операции. Операторы языка Java, как впрочем, и других языков программирования имеют свои назначения. Так существуют арифметические операторы, операторы инкремента и декремента, логические операторы и операторы отношения. Давайте рассмотри каждый из вышеупомянутых операторов.

Арифметические операторы

Все арифметические операторы очень просты и аналогичны операторам умножения «*», деления «/», сложения «+» и вычитания «–» используемые в математике. Существует оператор деления по модулю «%» и слегка запутанная на первый взгляд ситуация с оператором равно «=». Оператор равно в языках программирования называется оператором присваивания:

```
int x = 3
```

Здесь вы переменной x присваиваете значение 3. А оператор «равно» в языках программирования соответствует записи двух подряд операторов «равно»: «==». Рассмотрим на примере, что могут делать различные арифметические

операторы.

```
int x, y, z;
x = 5;
y = 3;
z = 0;
z = x + y;
```

В данном случае z будет иметь значение уже суммы x и y, то есть 8.

```
x = z*x;
```

Переменная x имела значение 5, но после такой записи предыдущее значение теряется и записывается произведение z*x (8*5), что равно 40. Теперь, если мы продолжим дальше наш код, то переменные будут иметь такой вид:

```
// x = 40;
// y = 3;
// z = 8;
```

Операторы сложения и вычитания имеют те же назначения что и в математике. Отрицательные числа так же родственны.

Операторы декремента «—» и инкремента «++» весьма специфичны, но очень просты. В программировании часто встречаются моменты, когда требуется увеличить или уменьшить значение на единицу. Часто это встречается в циклах. Операция инкремента увеличивает переменную на единицу.

```
int x = 5;
x++;
// Здесь x уже равен 6
```

Операция декремента уменьшает переменную на единицу.

```
int x = 5;
x--;
// x равен 4
```

Операции инкремента и декремента могут быть пост и префиксными:

```
int x = 5;
int y = 0;
y = x++;
```

В последней строке кода сначала значение x присваивается y, это значение 5, и только потом переменная x увеличивается на единицу. Получается что:

x = 6, y = 5

Префиксный инкремент имеет вид:

```
int x = 3;
int y = 0;
y = ++x;
```

И в этом случае, сначала переменная x увеличивается на один, а потом присваивает уже увеличенное значение y.

y = 4, x = 4

Операторы отношения

Операторы отношения позволяют проверить равенство обеих частей выражения. Имеется оператор равенства «==», операторы меньше «<» и больше «>», меньше или равно «<=» и больше или равно «>=», а так же оператор отрицания «!=».

```
9 == 10;
```

Это выражение не верно, девять не равно десяти, поэтому его значение этого выражения равно false.

```
9 != 10;
```

Здесь же, наоборот, оператор отрицания указывает на неравенство выражения, и значение будет равно true. Операторы больше, меньше, больше или равно и меньше или равно аналогичны соответствующим операторам из математики.

Логические операторы

Существует два логических оператора. Оператор «И», обозначающийся значками «&&» и оператор «ИЛИ», обозначенный в виде двух прямых слэшей «||». Например, имеется выражение:

```
A*B && B*C;
```

В том случае, если только обе части выражения истинны, значение выражения считается истинным. Если одна из частей

неверна, то значение всего выражения будет ложным. В противовес оператору «&&» имеется оператор «||», не напрасно имеющий название «ИЛИ».

A*B || B*C;

Если любая из частей выражения истинна, то и все выражение считается истинным. Оба оператора можно комбинировать в одном выражении, например:

A*B || B*C && C*D || B*A;

С помощью этого выражения я вас ввел, как мне кажется, в затруднение, неправда ли? Дело в том, что в Java, как и в математике существует приоритет или так называемая иерархия операторов, с помощью которой определяется какой из операторов главнее, а, следовательно, и проверяется первым. Рассмотрим с помощью списка приоритет всех имеющихся операторов языка Java:

[], ., (),
!, ~, ++, – –, + (унарный), – (унарный), new,
*, /, %,
+, –,
<<, >>, >>>,
<, <=, >, >=,
= =, !=,
&, ^, |,
&&,
||,
?:,
=, +=, –=, *=, /=, %=, |=, ^=, <<=, >>=, >>>=.

Ассоциативность операторов в списке следует слева направо и сверху вниз. То есть все, что находится левее и выше – старше по званию и главнее.

Вопросы и задания для самопроверки:

1. История создания языка.
2. Основные направления Java.

3. JDK и JRE.
4. Среды разработки для Java.
5. Синтаксис языка.
6. Типы данных.
7. Операторы.
8. Управляющие конструкции.
9. Массивы.
10. Абстракция.

Литература

Основная

1. Трофимов, В. В. Алгоритмизация и программирование : учебник для академического бакалавриата / В. В. Трофимов, Т. А. Павловская ; под ред. В. В. Трофимова. — М. : Издательство Юрайт, 2017. [Электронный ресурс] https://www.biblio-online.ru/viewer/B08DB966-3F96-4B5A-B030-E3CD9085CED4#page/1, 05.10.2017.

2. Соколова, В. В. Вычислительная техника и информационные технологии. Разработка мобильных приложений : учебное пособие для прикладного бакалавриата / В. В. Соколова. — М. : Издательство Юрайт, 2017 [Электронный ресурс] https://www.biblio-online.ru/viewer/D80F822D-BA6D-45E9-B83B-8EC049F5F7D9#page/1, 05.10.2017.

Дополнительная:

1. Васильев, Алексей Николаевич. Java**.** Объектно-ориентированное программирование для магистров и бакалавров [Текст]: базовый курс по объектно-ориентированному программированию : [учебное пособие] / А. Н. Васильев. - Санкт-Петербург [и др.] : Питер, 2014. - 396 с. - (Учебное пособие) (Стандарт третьего поколения). - Библиогр.: с. 377. - ISBN 978-5-496-00044-4

2. Эванс, Бенджамин. Java. Новое поколение разработки [Текст]: техники Java 7 и многоязычное программирование: [пер. с англ.] / Б. Эванс, М. Вербург. - Санкт-Петербург [и др.] :

Питер, 2014. - 556 с. : ил. - ISBN 978-1617290060. - ISBN 978-5-496-00544-9

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

1. Российское образование, федеральный портал [Официальный сайт] — URL: http://www.edu.ru
2. Образовательный портал «Учеба» [Официальный сайт] URL: http://www.ucheba.com/
3. Портал «Российское образование» [Официальный сайт] URL: http://www.edu.ru/

Тема 2 Разработка HTML-страниц

Содержание темы: Проектирование элементов оконного дизайна. Модель сообщений в Java. Программирования обработчиков сообщений.

Методические рекомендации

В станадарт Java входит два пакета для создания оконного пользовательского интерфейса: awt и его надстройка swing. Компоненты swing имеют расширинные возможности по сравнению с аналогичными awt компонентами, поэтому упор будет сделан на них. Во-вторых, в отличие от awt компоненты swing не содержат платформо-зависимого кода. Так сказать являются *облегченными (lightweight) компонентами*.

контейнеры верхнего уровня

- *JApplet* - главное окно апплета;
- *JFrame* - окно приложения;
- *JDialog* - диалог приложения.
- *JColorChooser* - диалог выбора цвета;
- *JFileChooser* - диалог выбора файлов и директорий;
- *FileDialog* - диалог выбора файлов и директорий (awt компонент).

простые контейнеры

- *JPanel* - простая панель для группировки элементов, включая вложенные панели;
- *JToolBar* - панель инструментов (обычно это кнопки);
- *JScroolPane* - панель прокрутки, позволяющая прокручивать содержимое дочернего элемента;
- *JDesktopPane* - контейнер для создания виртуального рабочего стола или приложений на основе MDI (multiple-document interface);
- *JEditorPane, JTextPane* - контейнеры для отображения сложного документа как HTML или RTF;

- *JTabbedPane* - контейнер для управления закладками;
- *JSplitPane* - контейнер, разделяющий два элемента и позволяющий пользователю изменять их размер.

элементы интерфейса

Следующие элементы управления могут использоваться и как контейнеры, так как наследуются от класса java.awt.Container.

- *JButton* - кнопка;
- *JCheckBox* - кнопка-флажок;
- *JComboBox* - выпадающий список;
- *JLabel* - метка, надпись;
- *JList* - список;
- *JPasswordField* - текстовое поле для скрытого ввода;
- *JProgressBar* - компонент для отображения числа в некотором диапазоне;
- *JRadioButton* - преключатели, радио-кнопки, обычно используется с компонентом ButtonGroup;
- *JSlider* - компонент позволяющий выбрать значение из заданного диапазона;
- *JSpinner* - компонент позволяющий выбрать значение из указанной последовательности;
- *JTable* - таблица;
- *JTextField* - однострочное текстовое поле;
- *JTextArea* - многострочное текстовое поле;
- *JTree* - дерево.

управление позиционированием и размерами

Для автоматического позиционирования и задания размеров дочерних элементов контейнеры используют специальные объекты - *компоновщики*. Для ручного позиционирования надо установить пустой объект вызовом метода setLayout(null).

Ниже приведен список стандартных компоновщиков:

- *BorderLayout* - размещает элементы в один из пяти регионов, как было указано при добавлении элемента в контейнер: наверх, вниз, влево, вправо, в центр;
- *FlowLayout* - размещает элементы по порядку в том же направлении, что и ориентация контейнера (слева на право по умолчанию) применяя один из пяти видов выравнивания, указанного при создании менеджера. Данный менеджер используется по умолчанию в большинстве контейнерах;
- *GridLayout* - размещает элементы таблично. Колличество столбцов и строк указывается при создании менеджера. По умолчанию одна строка, а число столбцов равно числу элементов;
- *BoxLayout* - размещает элементы по вертикали или по горизонтали. Обычно он используется не напрямую а через контейнерный класс Box, который имеет дополнительные возможности;
- *SpringLayout* - это менеджер низкого уровня и был разработан для программ построителей форм;
- *GroupLayout* - данный менеджер также был разработан для построителей форм.

события

Элементы интерфейса и контейнеры генерируют ряд событий, например:

- *ActionEvent* - событие, определяемое компонентом, например нажатие кнопки;
- *ComponentEvent* - генерируется, когда позиция, размер или видимость компонента изменяются;
- *ContainerEvent* - генерируется при добавлении или удалении элемента из контейнера;
- *FocusEvent* - генерируется при получении или потери фокуса ввода компонентом;
- *ItemEvent* - событие выбора или отменены выбора элемента. Например, изменение состояния кнопки-флажка, выбор элемента меню или списка;
- *KeyEvent* - событие ввода с клавиатуры;

- *MouseEvent* - события мыши;
- *WindowEvent* - события окна, как активация и свертывание.

Для обработки событий элемента в нем необходимо зарегистрировать объект обработчик в числе слушателей. Делается это методами типа addxxxListener, например addMouseListener(). В качестве объектов обработчиков обычно выбирают контейнер, в котором содержится элемент. Обработка события осуществляется через соответствующий интерфейс, например:

- *ActionListener* - интерфейс обработки события ActionEvent;
- *ItemListener* - интерфейс обработки события ItemEvent;
- *KeyListener* - интерфейс обработки события KeyEvent;
- *MouseListener* - интерфейс обработки события MouseEvent, для нажатия/отжатия кнопок и входа/ухода курсора мыши с области компонента;
- *MouseMotionListener* - интерфейс обработки события MouseEvent, для движение курсора мыши или перетаскивании мышью;
- *MouseWheelListener* - интерфейс обработки события MouseEvent, для прокрутки колеса мыши.

Если весь интерфейс реализовывать не нужно, например, нужен только один из методов, то можно воспользоваться адаптерами.

Вопросы и задания для самопроверки:

1. Объекты.
2. Абстрактные классы.
3. Интерфейсы.
4. Управление доступом.
5. Инкапсуляция.

6. Наследование и полиморфизм.
7. Коллекции объектов.
8. Обработка ошибок и исключения.
9. Внутренние и анонимные (безымянные) внутренние классы.
10. Система ввода-вывода Java.

Литература

Основная

1. Соколова, В. В. Вычислительная техника и информационные технологии. Разработка мобильных приложений : учебное пособие для прикладного бакалавриата / В. В. Соколова. — М. : Издательство Юрайт, 2017 [Электронный ресурс] https://www.biblio-online.ru/viewer/D80F822D-BA6D-45E9-B83B-8EC049F5F7D9#page/1, 05.10.2017.

2. Тузовский, А. Ф. Объектно-ориентированное программирование : учебное пособие для прикладного бакалавриата / А. Ф. Тузовский. — М. : Издательство Юрайт, 2017. [Электронный ресурс] https://www.biblio-online.ru/viewer/BDEEFB2D-532D-4306-829E-5869F6BDA5F9#page/1, 05.10.2017.

Дополнительная:

1. Баженова, И.Ю. Язык программирования Java [Электронный ресурс] / И.Ю. Баженова. - Москва : Диалог-МИФИ, 2008. - 254 с. : табл., ил. - URL: http://biblioclub.ru/index.php?page=book&id=54745

2. Кулямин, В. Компонентный подход в программировании [Электронный ресурс]/ В. Кулямин. - 2-е изд., исправ. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 591 с. : ил. - (Основы информационных технологий). - Библиогр. в кн. - URL: http://biblioclub.ru/index.php?page=book&id=429086

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

1. Единое окно доступа к образовательным ресурсам «Единое окно» [Официальный сайт] URL: http://window.edu.ru/
2. Федеральная университетская компьютерная сеть России [Официальный сайт] URL: http://www.runnet.ru/
3. Служба тематических толковых словарей[Официальный сайт] URL: http://www.glossary.ru/

Тема 3 Язык программирования JavaScript. Динамический HTML

Содержание темы: Программирование апплетов. Программирование многопоточных приложений. Программирование сервлетов.

Методические рекомендации

Когда число компьютеров в учреждении переваливает за десяток и сотрудникам надоедает бегать с дискетами друг к другу для обмена файлами, тогда в компьютеры вставляются сетевые карты, протягиваются кабели и компьютеры объединяются в сеть. Сначала все компьютеры в сети равноправны, они делают одно и то же — это *одноранговая* (peer-to-peer) сеть. Потом покупается компьютер с большими и быстрыми жесткими дисками, и все файлы учреждения начинают храниться на данных дисках — этот компьютер становится файл-сервером, предоставляющим услуги хранения, поиска, архивирования файлов. Затем покупается дорогой и быстрый принтер. Компьютер, связанный с ним, становится принт-сервером, предоставляющим услуги печати. Потом появляются графический сервер, вычислительный сервер, сервер базы данных. Остальные компьютеры становятся клиентами этих серверов. Такая архитектура сети называется архитектурой *клиент-сервер* (client-server). Сервер постоянно находится в состоянии ожидания, он *прослушивает* (listen) сеть, ожидая запросов от клиентов. Клиент связывается с сервером и посылает ему *запрос* (request) с описанием услуги, например, имя нужного файла. Сервер обрабатывает запрос и отправляет ответ (response), в нашем примере, файл, или сообщение о невозможности оказать услугу. После этого связь может быть разорвана или продолжиться, организуя сеанс связи, называемый *сессией* (session).

Запросы клиента и ответы сервера формируются по строгим правилам, совокупность которых образует *протокол* (protocol) связи. Всякий протокол должен, прежде всего, содержать правила соединения компьютеров. Клиент перед

посылкой запроса должен удостовериться, что сервер в рабочем состоянии, прослушивает сеть, и услышал клиента. Послав запрос, клиент должен быть уверен, что запрос дошел до сервера, сервер понял запрос и готов ответить на него. Сервер обязан убедиться, что ответ дошел до клиента. Окончание сессии должно быть четко зафиксировано, чтобы сервер мог освободить ресурсы, занятые обработкой запросов клиента.

Все правила, образующие протокол, должны быть понятными, однозначными и короткими, чтобы не загружать сеть. Поэтому сообщения, пересылаемые по сети, напоминают шифровки, в них имеет значение каждый бит.

Итак, все сетевые соединения основаны на трех основных понятиях: клиент, сервер и протокол. Клиент и сервер — понятия относительные. В одной сессии компьютер может быть сервером, а в другой — клиентом. Например, файл-сервер может послать принт-серверу файл на печать, становясь его клиентом.

Для обслуживания протокола: формирования запросов и ответов, проверок их соответствия протоколу, расшифровки сообщений, связи с сетевыми устройствами создается программа, состоящая из двух частей. Одна часть программы работает на сервере, другая — на клиенте. Эти части так и называются серверной частью программы и клиентской частью программы, или, короче, сервером и клиентом.

Очень часто клиентская и серверная части программы пишутся отдельно, разными фирмами, поскольку от этих программ требуется только, чтобы они соблюдали протокол. Более того, по каждому протоколу работают десятки клиентов и серверов, отличающихся разными удобствами.

Обычно на одном компьютере-сервере работают несколько программ-серверов. Одна программа занимается электронной почтой, другая — пересылкой файлов, третья предоставляет Web-страницы. Для того чтобы их различать, каждой программе-серверу придается *номер порта* (port). Это просто целое положительное число, которое указывает клиент, обращаясь к определенной программе-серверу. Число, вообще

говоря, может быть любым, но наиболее распространенным протоколам даются стандартные номера, чтобы клиенты были твердо уверены, что обращаются к нужному серверу. Так, стандартный номер порта электронной почты 25, пересылки файлов — 21, Web-сервера — 80. Стандартные номера простираются от 0 до 1023. Числа, начиная с 1024 до 65 535, можно использовать для своих собственных номеров портов.

Все это похоже на телевизионные каналы. Клиент-телевизор обращается посредством антенны к серверу-телецентру и выбирает номер канала. Он уверен, что на первом канале ОРТ, на втором — РТР и т. д.

Чтобы равномерно распределить нагрузку на сервер, часто несколько портов прослушиваются программами-серверами одного типа. Web-сервер, кроме порта с номером 80, может прослушивать порт 8080, 8001 и еще какой-нибудь другой.

В процессе передачи сообщения используется несколько протоколов. Даже когда мы отправляем письмо, мы сначала пишем сообщение, начиная его: "Глубокоуважаемый Иван Петрович!" и заканчивая: "Искренне преданный Вам". Это один протокол. Можно начать письмо словами: "Вася, привет!" и закончить: "Ну, пока". Это другой протокол. Потом мы помещаем письмо в конверт и пишем на нем адрес по протоколу, предложенному Министерством связи. Затем письмо попадает на почту, упаковывается в мешок, на котором пишется адрес по протоколу почтовой связи. Мешок загружается в самолет, который перемещается по своему протоколу. Заметьте, что каждый протокол только добавляет к сообщению свою информацию, не меняя его, ничего не зная о том, что сделано по предыдущему протоколу и что будет сделано по правилам следующего протокола. Это очень удобно — можно программировать один протокол, ничего не зная о других протоколах.

Прежде чем дойти до адресата, письмо проходит обратный путь: вынимается из самолета, затем из мешка, потом

из конверта. Поэтому говорят о стеке (stack) протоколов: "Первым пришел, последним ушел".

В современных глобальных сетях принят стек из четырех протоколов, называемый стеком протоколов TCP/IP.

Сначала мы пишем сообщение, пользуясь программой, реализующей *прикладной* (application) протокол: HTTP (80), SMTP (25), TELNET (23), FTP (21), POP3 (100) или другой протокол. В скобках записан стандартный номер порта.

Затем сообщение обрабатывается по *транспортному* (transport) протоколу. К нему добавляются, в частности, номера портов отправителя и получателя, контрольная сумма и длина сообщения. Наиболее распространены транспортные протоколы TCP (Transmission Control Protocol) и UDP (User Datagram Protocol). В результате работы протокола TCP получается *TCP-пакет* (packet), а протокола UDP — *дейтаграмма* (datagram).

Дейтаграмма невелика — всего около килобайта, поэтому сообщение делится на прикладном уровне на части, из которых создаются отдельные дейтаграммы. Дейтаграммы посылаются одна за другой. Они могут идти к получателю разными маршрутами, прийти совсем в другом порядке, некоторые дейтаграммы могут потеряться. Прикладная программа получателя должна сама позаботиться о том, чтобы собрать из полученных дейтаграмм исходное сообщение. Для этого обычно перед посылкой части сообщения нумеруются, как страницы в книге. Таким образом, протокол UDP работает как почтовая служба. Посылая книгу, мы разрезаем ее на страницы, каждую страницу отправляем в своем конверте, и никогда не уверены, что все письма дойдут до адресата.

TCP-пакет тоже невелик, и пересылка также идет отдельными пакетами, но протокол TCP обеспечивает надежную связь. Сначала устанавливается соединение с получателем. Только после этого посылаются пакеты. Получение каждого пакета подтверждается получателем, при ошибке посылка пакета повторяется. Сообщение аккуратно собирается получателем. Для отправителя и получателя создается впечатление, что

пересылаются не пакеты, а сплошной поток байтов, поэтому передачу сообщений по протоколу TCP часто называют передачей потоком. Связь по протоколу TCP больше напоминает телефонный разговор, чем почтовую связь.

Далее сообщением занимается программа, реализующая *сетевой* (network) протокол. Чаще всего это протокол IP (Internet Protocol). Он добавляет к сообщению адрес отправителя и адрес получателя, и другие сведения. В результате получается *IP-пакет.*

Наконец, IP-пакет поступает к программе, работающей по *канальному* (link) протоколу ENET, SLIP, PPP, и сообщение принимает вид, пригодный для передачи по сети.

На стороне получателя сообщение проходит через эти четыре уровня протоколов в обратном порядке, освобождаясь от служебной информации, и доходит до программы, реализующей прикладной протокол.

Какой же адрес заносится в IP-пакет? Каждый компьютер или другое устройство, подключенное к объединению сетей Internet, так называемый *хост* (host), получает уникальный номер — четырехбайтовое целое число, называемое *IP-адресом* (IP-address). По традиции содержимое каждого байта записывается десятичным числом от 0 до 255, называемым *октетом* (octet), и эти числа пишутся через точку: 138.2.45.12 или 17.056.215.38.

IP-адрес удобен для машины, но неудобен для человека. Представьте себе рекламный призыв: "Заходите на наш сайт 154.223.145.26!" Поэтому IP-адрес хоста дублируется *доменным именем* (domain name).

В доменном имени присутствует краткое обозначение страны: ru — Россия, su — Советский Союз, ua — Украина, de — ФРГ и т. д., или обозначение типа учреждения: com — коммерческая структура, org — общественная организация, edu — образовательное учреждение. Далее указывается регион: msc.ru — Москва, spb.ru — Санкт-Петербург, kcn.ru — Казань, или учреждение: bhv.ru — "БХВ-Петербург", ksu.ru —

Казанский госуниверситет, sun.com — SUN Microsystems. Потом подразделение: www.bhv.ru, java.sun.com. Такую цепочку кратких обозначений можно продолжать и дальше.

В Java IP-адрес и доменное имя объединяются в один класс inetAddress пакета java.net. Экземпляр этого класса создается статическим методом getByName (string host) данного же класса, в котором аргумент host— это доменное имя или IP-адрес.

Работа в WWW

Среди программного обеспечения Internet большое распространение получила информационная система WWW (World Wide Web), основанная на прикладном протоколе HTTP (Hypertext Transfer Protocol). В ней используется расширенная адресация, называемая URL (Uniform Resource Locator). Эта адресация имеет такие схемы:

protocol://authority@host:port/path/file#ref

protocol://authority@host:port/path/file/extra_path?info

Здесь необязательная часть authority — это пара имя:пароль для доступа к хосту, host — это IP-адрес или доменное имя хоста. Например:

http://www.bhv.ru/

http://132.192.5.10:8080/public/some.html

ftp://guest:password@lenta.ru/users/local/pub

ffle:///C:/text/html/index.htm

Если какой-то элемент URL отсутствует, то берется стандартное значение. Например, в первом примере номер порта port равен 80, а имя файла path — какой-то головной файл, определяемый хостом, чаще всего это файл с именем index.html. В третьем примере номер порта равен 21. В последнем примере в форме URL просто записано имя файла index.htm, расположенного на разделе C: жесткого диска той же самой машины.

В Java для работы с URL есть класс URL пакета java.net. Объект этого класса создается одним из шести конструкторов. В основном конструкторе

URL(String url)

задается расширенный адрес url в виде строки. Кроме методов доступа getxxxo, позволяющих получить элементы URL, в этом классе есть два интересных метода:

- openConnection () — определяет связь с URL и возвращает объект класса URLConnection;
- openStream() — устанавливает связь с URL и открывает входной поток в виде возвращаемого объекта класса inputstream.

Листинг 19.1 показывает, как легко можно получить файл из Internet, пользуясь методом openStream().

Листинг 19.1. Получение Web-страницы

```
import java.net.*;
import j ava.io.*;
class SimpleURL{
public static void main(String[] args){
try{
URL bhv = new URL("http://www.bhv.ru/");
BufferedReader br = new BufferedReader(
new InputStreamReader(bhv.openStream()));
String line;
while ((line = br.readLine()) != null)
System.out.println(line);
br.close();
}catch(MalformedURLException me){
System.err.println("Unknown host: " + me);
System.exit(0);
}catch(IOException ioe){
System.err.println("Input error: " + ioe);
}
}
}
```

Если вам надо не только получить информацию с хоста, но и узнать ее тип: текст, гипертекст, архивный файл, изображение, звук, или выяснить длину файла, или передать

информацию на хост, то необходимо сначала методом openConnection () создать объект класса URLConnection или его подкласса HttpURLConnection.

После создания объекта соединение еще не установлено, и можно задать параметры связи. Это делается следующими методами:

- setDoOutput (boolean out) — если аргумент out равен true, то передача пойдет от клиента на хост; значение по умолчанию false;
- setDoinput (boolean in) — если аргумент in равен true, то передача пойдет с хоста к клиенту; значение по умолчанию true, но если уже выполнено setDoOutput(true), то значение по умолчанию равно false;
- setUseCaches (boolean cache) — если аргумент cache равен false, то передача пойдет без кэширования, если true, то принимается режим по умолчанию;
- setDefaultUseCaches(boolean default) — если аргумент default равен true, то принимается режим кэширования, предусмотренный протоколом;
- setRequestProperty(String name, String value) — добавляет параметр name со значением value к заголовку посылаемого сообщения.

После задания параметров нужно установить соединение методом connect (). После соединения задание параметров уже невозможно. Следует учесть, что некоторые методы доступа getxxxo, которым надо получить свои значения с хоста, автоматически устанавливают соединение, и обращение к методу connect () становится излишним.

Web-сервер возвращает информацию, запрошенную клиентом, вместе с заголовком, сведения из которого можно получить методами getxxxo, например:

- getcontentType () — возвращает строку типа string, показывающую тип пересланной информации, например, "text/html", или null, если сервер его не указал;

- getcontentLength () — возвращает длину полученной информации в байтах или — 1, если сервер ее не указал;
- getcontent () — возвращает полученную информацию в виде объекта типа Object;
- getContentEncoding () — возвращает строку типа string с кодировкой полученной информации, или null, если сервер ее не указал.

Два метода возвращают потоки ввода/вывода, созданные для данного соединения:

- getlnputStream() — возвращает входной поток типа InputStream;
- getOutputStream() — возвращает выходной поток типа OutputStream.

Прочие методы, а их около двадцати, возвращают различные параметры соединения.

Обращение к методу bhv.openstreamo, записанное в листинге 19.1, — это, на самом деле, сокращение записи

bhv.openConnection().getlnputStream()

В листинге 19.2 показано, как переслать строку текста по адресу URL.

Web-сервер, который получает эту строку, не знает, что делать с полученной информацией. Занести ее в файл? Но с каким именем, и есть ли у него право создавать файлы? Переслать на другую машину? Но куда?

Выход был найден в системе CGI (Common Gateway Interface), которая вкратце действует следующим образом. При посылке сообщения мы указываем URL исполнимого файла некоторой программы, размещенной на машине-сервере. Получив сообщение, Web-сервер запускает эту программу и передает сообщение на ее стандартный ввод. Вот программа-то и знает, что делать с полученным сообщением. Она обрабатывает сообщение и выводит результат обработки на свой стандартный вывод. Web-сервер подключается к стандартному выводу, принимает результат и отправляет его обратно клиенту.

CGI-программу можно написать на любом языке: C, C++, Pascal, Perl, PHP, лишь бы у нее был стандартный ввод и стандартный вывод. Можно написать ее и на Java, но в технологии Java есть более изящное решение этой задачи с помощью сервлетов (servlets). CGI-программы обычно лежат на сервере в каталоге cgi-bin.

Листинг 19.2. Посылка строки по адресу URL

```
import java.net.*;
import java.io.*;
class PostURL{
public static void main(String[] args){
String req = "This text is posting to URL";
try{
// Указываем URL нужной CGI-программы
URL url = new URL("http://www.bhv.ru/cgi-bin/some.pl");
// Создаем объект uc
URLConnection uc = url.openConnection();
// Собираемся отправлять
uc.setDoOutput(true);
// и получать сообщения
uc.setDoInput(true);
// без кэширования
uc.setUseCaches(false);
// Задаем тип
uc.setRequestProperty("content-type",
"application/octet-stream");
// и длину сообщения
uc.setRequestProperty("content-length", "" + req.length());
// Устанавливаем соединение
uc.connect();
// Открываем выходной поток
DataOutputStream dos = new DataOutputStream(
uc.getOutputStreamO);
// и выводим в него сообщение, посылая его на адрес
URL dos.writeBytes(req);
```

```
// Закрываем выходной поток
dos.close();
// Открываем входной поток для ответа сервера
BufferedReader br = new BufferedReader(new
InputStreamReader(
uc.getlnputStream() )) ;
String res = null;
// Читаем ответ сервера и выводим его на консоль
while ((res = br.readLine()) != null)
System.out.println(res);
br.close () ;
}catch(MalformedURLException me){
System.err.println(me);
}catch(UnknownHostException he){
System.err.println(he);
}catch(UnknownServiceException se){
System.err.println(se);
}catch(IOException ioe){
System.err.println(ioe);
}
}
}
```

Работа по протоколу TCP

Программы-серверы, прослушивающие свои порты, работают под управлением операционной системы. У машин-серверов могут быть самые разные операционные системы, особенности которых передаются программам-серверам.

Чтобы сгладить различия в реализациях разных серверов, между сервером и портом введен промежуточный программный слой, названный *сокетом* (socket). Английское слово socket переводится как электрический разъем, розетка. Так же как к розетке при помощи вилки можно подключить любой электрический прибор, лишь бы он был рассчитан на 220 В и 50 Гц, к соке-ту можно присоединить любой клиент, лишь бы он работал по тому же протоколу, что и сервер. Каждый сокет

связан (bind) с одним портом, говорят, что сокет прослушивает (listen) порт. Соединение с помощью сокетов устанавливается так.

1. Сервер создает сокет, прослушивающий порт сервера.

2. Клиент тоже создает сокет, через который связывается с сервером, сервер начинает устанавливать (accept) связь с клиентом.

3. Устанавливая связь, сервер создает новый сокет, прослушивающий порт с другим, новым номером, и сообщает этот номер клиенту.

4. Клиент посылает запрос на сервер через порт с новым номером.

После этого соединение становится совершенно симметричным — два сокета обмениваются информацией, а сервер через старый сокет продолжает прослушивать прежний порт, ожидая следующего клиента.

В Java сокет — это объект класса socket из пакета java.io. В классе шесть конструкторов, в которые разными способами заносится адрес хоста и номер порта. Чаще всего применяется конструктор

Socket(String host, int port)

Многочисленные методы доступа устанавливают и получают параметры сокета. Мы не будем углубляться в их изучение. Нам понадобятся только методы, создающие потоки ввода/вывода:

- getlnputStream() — возвращает входной поток типа InputStream;
- getOutputStream() — возвращает выходной поток типа OutputStream.

Приведем пример получения файла с сервера по максимально упрощенному протоколу HTTP.

1. Клиент посылает серверу запрос на получение файла строкой "POST filename HTTP/1.l\n\n", где filename — строка с путем к файлу на сервере.

2. Сервер анализирует строку, отыскивает файл с именем filename и возвращает его клиенту. Если имя файла filename заканчивается наклонной чертой /, то сервер понимает его как имя каталога и возвращает файл in-dex.html, находящийся в этом каталоге.

3. Перед содержимым файла сервер посылает строку вида "HTTP/1.1 code OK\n\n", где code — это код ответа, одно из чисел: 200 — запрос удовлетворен, файл посылается; 400 — запрос не понят; 404 — файл не найден.

4. Сервер закрывает сокет и продолжает слушать порт, ожидая следующего запроса.

5. Клиент выводит содержимое полученного файла в стандартный вывод System, out или выводит код сообщения сервера в стандартный вывод сообщений System, err.

6. Клиент закрывает сокет, завершая связь.

Этот протокол реализуется в клиентской программе листинга 19.3 и серверной программе листинга 19.4.

Листинг 19.3. Упрощенный HTTP-клиент

```
import java.net.*;
import java.io.*;
import java.util.*;
class Client{
public static void main(String[] args){
if (args.length != 3){
System.err.println("Usage: Client host port file");
System.exit(0) ;
}
String host = args[0];
int port = Integer.parselnt(args[1]);
String file = args[2];
try{
Socket sock = new Socket(host, port);
PrintWriter pw = new PrintWriter(new OutputStreamWriter(
sock.getOutputStreamf)), true);
pw.println("POST " + file + " HTTP/1.l\n");
```

```
        BufferedReader br = new BufferedReader(new
InputStreamReader(
        sock.getlnputStream() ) ) ;
        String line = null;
        line = br.readLine();
        StringTokenizer st = new StringTokenizer(line);
        String code = null;
        if ((st.countTokens() >= 2) &&
st.nextToken().equals("POST")){
        if ((code = st.nextToken()) != "200") {
        System.err.println("File not found, code = " + code);
        System.exit (0);
        }
        }
        while ((line = br.readLine()) != null)
        System.out.println{line);
        sock.close();
        }catch(Exception e){
        System.err.println(e);
        }
        }
        }
```

Закрытие потоков ввода/вывода вызывает закрытие сокета. Обратно, закрытие сокета закрывает и потоки.

Для создания сервера в пакете java.net есть класс serversocket. В конструкторе этого класса указывается номер порта

ServerSocket(int port)

Основной метод этого класса accept () ожидает поступления запроса. Когда запрос получен, метод устанавливает соединение с клиентом и возвращает объект класса socket, через который сервер будет обмениваться информацией с клиентом.

Листинг 19.4. Упрощенный HTTP-сервер

```
import j ava.net.*;
import java.io.*;
import j ava.uti1.*;
class Server!
public static void main(String[] args){
try{
ServerSocket ss = new
ServerSocket(Integer.parselnt(args[0]));
while (true)
new HttpConnect(ss.accept());
}catch(ArraylndexOutOfBoundsException ae){
System.err.println("Usage: Server port");
System.exit(0);
}catch(IOException e){
System.out.println(e);
}
}
}
class HttpConnect extends Thread{
private Socket sock;
HttpConnect(Socket s) {
sock = s;
setPriority(NORM_PRIORITY - 1);
start {) ;
}
public void run(){
try{
PrintWriter pw = new PrintWriter(new OutputStreamWriter(
sock.getOutputStream()}, true);
BufferedReader br = new BufferedReader(new
InputStreamReader(
sock.getlnputStream() ) ) ;
String req = br.readLine();
System.out.println("Request: " + req);
StringTokenizer st = new StringTokenizer(req);
```

```
if ((st.countTokens() >= 2) &&
st.nextToken().equals("POST")){
if ((req = st.nextToken()).endsWith("/") II req.equals(""))
req += "index.html";
try{
File f = new File(req);
BufferedReader bfr =
new BufferedReader(new FileReader(f));
char[] data = new char[(int)f.length()];
bfr.read(data);
pw.println("HTTP/1.1 200 OK\n");
pw.write(data);
pw.flush();
}catch(FileNotFoundException fe){
pw.println("HTTP/1.1 404 Not FoundXn");
}catch(lOException ioe){
System.out.println(ioe);
}
}else pw.println("HTTP/l.l 400 Bad RequestW);
sock.close();
}catch(IOException e){
System.out.println(e);
}
}
}
```

Вначале следует запустить сервер, указав номер порта, например:

Java Server 8080

Затем надо запустить клиент, указав IP-адрес или доменное имя хоста, номер порта и имя файла:

Java Client localhost 8080 Server.Java

Сервер отыскивает файл Server.java в своем текущем каталоге и посылает его клиенту. Клиент выводит содержимое этого класса в стандартный вывод и завершает работу. Сервер продолжает работать, ожидая следующего запроса.

Замечание по отладке

Программы, реализующие стек протоколов TCP/IP, всегда создают так называемую "петлю" с адресом 127.0.0.1 и доменным именем localhost. Это адрес самого компьютера. Он используется для отладки приложений клиент-сервер. Вы можете запускать клиент и сервер на одной машине, пользуясь этим адресом.

Работа по протоколу UDP

Для посылки дейтаграмм отправитель и получатель создают сокеты дейта-граммного типа. В Java их представляет класс DatagramSocket. В классе три конструктора:

- DatagramSocket () — создаваемый сокет присоединяется к любому свободному порту на локальной машине;
- DatagramSocket (int port) — создаваемый сокет присоединяется к порту port на локальной машине;
- DatagramSocket(int port, InetAddress addr) — создаваемый СОКСТ присоединяется к порту port; аргумент addr — один из адресов локальной машины.

Класс содержит массу методов доступа к параметрам сокета и, кроме того, методы отправки и приема дейтаграмм:

- send(DatagramPacket pack) — отправляет дейтаграмму, упакованную в пакет pack;
- receive (DatagramPacket pack) — дожидается получения дейтаграммы и заносит ее в пакет pack.

При обмене дейтаграммами соединение обычно не устанавливается, дейтаграммы посылаются наудачу, в расчете на то, что получатель ожидает их. Но можно установить соединение методом

connect(InetAddress addr, int port)

При этом устанавливается только одностороннее соединение с хостом по адресу addr и номером порта port — или на отправку или на прием дейтаграмм. Потом соединение можно разорвать методом

disconnect()

При посылке дейтаграммы по протоколу JJDP сначала создается сообщение в виде массива байтов, например,

String mes = "This is the sending message.";

byte[] data = mes.getBytes();

Потом записывается адрес — объект класса inetAddress, например:

InetAddress addr = InetAddress.getByName (host);

Затем сообщение упаковывается в пакет — объект класса DatagramPacket. При этом указывается массив данных, его длина, адрес и номер порта:

DatagramPacket pack = new DatagramPacket(data, data.length, addr, port)

Далее создается дейтаграммный сокет

DatagramSocket ds = new DatagramSocket()

и дейтаграмма отправляется

ds.send(pack)

После посылки всех дейтаграмм сокет закрывается, не дожидаясь какой-либо реакции со стороны получателя:

ds.close ()

Прием и распаковка дейтаграмм производится в обратном порядке, вместо метода send () применяется метод receive (DatagramPacket pack).

В листинге 19.5 показан пример класса Sender, посылающего сообщения, набираемые в командной строке, на localhost, порт номер 1050. Класс Recipient, описанный в листинге 19.6, принимает эти сообщения и выводит их в свой стандартный вывод.

Листинг 19.5. Посылка дейтаграмм по протоколу UDP

```
import java.net.*;
import java.io.*;
class Sender{
private String host;
private int port;
Sender(String host, int port){
this.host = host;
```

```
this.port = port;
}
private void sendMessage(String mes){
try{
byte[] data = mes.getBytes();
InetAddress addr = InetAddress.getByName(host);
DatagramPacket pack =
new DatagramPacket(data, data.length, addr, port);
DatagramSocket ds = new DatagramSocket();
ds.send(pack);
ds.close();
}catch(IOException e){
System.err.println(e);
}
}
public static void main(String[] args){
Sender sndr = new Sender("localhost", 1050);
for (int k = 0; k < args.length; k++)
sndr.sendMessage(args[k]);
}
}
```

Листинг 19.6. Прием дейтаграмм по протоколу UDP

```
import j ava.net.*;
import java.io.*;
class Recipient{
public static void main(String[] args)(
try{
DatagramSocket ds = new DatagramSocket(1050);
while (true){
DatagramPacket pack =
new DatagramPacket(new byte[1024], 1024);
ds.receive(pack);
System.out.println(new String(pack.getData()));
}
)catch(Exception e){
```

```
System.out.println(e);
}
}
}
```

Вопросы и задания для самопроверки:

1. Интерфейсы Observable, Iterable, Comparable, Cloneable
2. События и их слушатели (ActionListeners)
3. Библиотека Swing.
4. Model-View-Controller.
5. Диспетчеры компоновки.
6. Библиотека SWT.
7. Работа с сетевыми протоколами.
8. Интернационализация.
9. Работа с базами данных.
10. Работа со звуком и графикой.
11. Удаленный вызов методов.

Литература

Основная

1. Трофимов, В. В. Алгоритмизация и программирование : учебник для академического бакалавриата / В. В. Трофимов, Т. А. Павловская ; под ред. В. В. Трофимова. — М. : Издательство Юрайт, 2017. [Электронный ресурс] https://www.biblio-online.ru/viewer/B08DB966-3F96-4B5A-B030-E3CD9085CED4#page/1, 05.10.2017.
2. Соколова, В. В. Вычислительная техника и информационные технологии. Разработка мобильных приложений : учебное пособие для прикладного бакалавриата / В. В. Соколова. — М. : Издательство Юрайт, 2017 [Электронный ресурс] https://www.biblio-online.ru/viewer/D80F822D-BA6D-45E9-B83B-8EC049F5F7D9#page/1, 05.10.2017.

Дополнительная:

1. Кулямин, В. Компонентный подход в программировании [Электронный ресурс]/ В. Кулямин. - 2-е изд., исправ. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 591 с. : ил. - (Основы информационных технологий). - Библиогр. в кн. - URL: http://biblioclub.ru/index.php?page=book&id=429086

2. Истомин, Евгений Петрович.Высокоуровневые методы информатики и программирования [Текст] : учебник для студентов вузов / Е. П. Истомин, В. В. Новиков, М. В. Новикова ; Рос. гос. гидрометеоролог. ун-т. - Изд. 3-е. - СПб. : Андреевский издательский дом, 2010. - 228 с.

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

1. Образовательный портал [Официальный сайт] URL: «Академик» http://dic.academic.ru/
2. Web of Sciense (архив с 2002 года) рефераты [Официальный сайт] URL: http://webofknowledge.com.
3. Лекториум “(Минобрнауки РФ) единая Интернет-библиотека лекций [Официальный сайт] URL http://www.lektorium.tv/

4. Перечень примерных вопросов для подготовки к зачету

Перечень вопросов к зачету

12. История создания языка.
13. Основные направления Java.
14. JDK и JRE.
15. Среды разработки для Java.
16. Синтаксис языка.
17. Типы данных.
18. Операторы.
19. Управляющие конструкции.
20. Массивы.

21. Абстракция.
22. Объекты.
23. Абстрактные классы.
24. Интерфейсы.
25. Управление доступом.
26. Инкапсуляция.
27. Наследование и полиморфизм.
28. Коллекции объектов.
29. Обработка ошибок и исключения.
30. Внутренние и анонимные (безымянные) внутренние классы.
31. Система ввода-вывода Java.
32. Интерфейсы Observable, Iterable, Comparable, Cloneable
33. События и их слушатели (ActionListeners)
34. Библиотека Swing.
35. Model-View-Controller.
36. Диспетчеры компоновки.
37. Библиотека SWT.
38. Работа с сетевыми протоколами.
39. Интернационализация.
40. Работа с базами данных.
41. Работа со звуком и графикой.
42. Удаленный вызов методов.