

Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

В.Г. Резник

РАСПРЕДЕЛЁННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Методические указания по выполнению лабораторных работ

Томск
2024

УДК 004.75
ББК 30.2-5-05
Р-344

Резник, Виталий Григорьевич

Р-344 Распределённые вычислительные системы. Методические указания по выполнению лабораторных работ / В.Г. Резник. – Томск : Томск. гос. ун-т систем упр. и радиоэлектроники, 2024. – 116 с.

Методические указания предназначены для проведения лабораторных работ по дисциплине «Распределённые вычислительные системы» для студентов направления подготовки бакалавра: 09.03.01 «Информатика и вычислительная техника».

Одобрено на заседании каф. АСУ протокол №_____ от _____

УДК 004.75
ББК 30.2-5-05

© Резник В. Г., 2024
© Томск. гос. ун-т систем упр. и радиоэлектроники, 2024

Оглавление

ВВЕДЕНИЕ.....	5
1 ВВЕДЕНИЕ В ТЕОРИЮ ВЫЧИСЛИТЕЛЬНЫХ СЕТЕЙ.....	6
1.1 Лабораторная работа №1. Тестирование ПО рабочей области студента.....	7
1.1.1 Общая структура ПО для проведения лабораторных работ.....	7
1.1.2 Рабочий стол пользователя upk и инструменты его рабочей области.....	8
1.1.3 Назначение каталогов рабочей области студента.....	9
1.1.4 Подключение инструментальных средств к рабочей среде студента.....	10
1.2 Лабораторная работа №2. Инструментальные средства реализации распределённых систем.....	12
1.2.1 Тестирование ПО языка Java.....	12
1.2.2 Тестирование ПО СУБД Apache Derby.....	16
1.2.3 Тестирование ПО сервера приложений Apache TomEE.....	20
1.2.4 Тестирование ПО среды разработки Eclipse EE.....	25
2 ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ЯЗЫКА JAVA.....	29
2.1 Лабораторная работа №3. Базовые инструментальные средства языка Java.....	30
2.1.1 Организация командной среды разработки проекта proj1.....	30
2.1.2 Компиляция и тестирование приложения Example1.....	34
2.1.3 Создание и запуск проекта proj1.jar.....	37
2.1.4 Реализация проекта proj1 в инструментальной среде IDE Eclipse EE.....	40
2.2 Лабораторная работа №4. Классы, интерфейсы и методы языка Java.....	48
2.2.1 Классы, объекты и объектные переменные языка Java.....	49
2.2.2 Объекты типа String и стандартный вывод информации.....	52
2.2.3 Преобразования простых типов данных.....	55
2.2.4 Использование массивов простых и объектных типов данных.....	57
2.2.5 Работа со строками данных типов String и StringBuffer.....	59
2.3 Лабораторная работа №5. Ввод/вывод языка Java.....	63
2.3.1 Классы и методы стандартного ввода/вывода.....	63
2.3.2 Байтовые потоки ввода/вывода средствами классов InputStream и OutputStream.....	67
2.3.3 Инструментальные средства класса File.....	69
2.3.4 Сериализация объектов и объектный ввод/вывод инструментальными средствами классов ObjectInputStream и ObjectOutputStream.....	73
2.3.5 Символьный ввод/вывод средствами классов Reader, Writer.....	78
2.4 Лабораторная работа №6. Сокеты и сетевое ПО языка Java.....	82
2.4.1 Реализация сетевой адресации объектами классов InetAddress, URL и URLConnection.....	82
2.4.2 Реализация сетевого приложения в виде класса TCPServer.....	85
2.4.3 Реализация сетевого приложения в виде класса TCPClient.....	90
2.5 Лабораторная работа №7. Технология работы с СУБД Apache Derby.....	95
2.5.1 Изучение инфраструктуры размещения СУБД Apache Derby.....	95
2.5.2 Создание учебной базы данных exampleDB.....	99
2.5.3 Реализация учебного примера Example11.....	105
3 ОБЪЕКТНЫЕ РАСПРЕДЕЛЁННЫЕ СИСТЕМЫ.....	113
3.1 Лабораторная работа №8. Программное проектирование распределённой системы.....	113

3.2 Лабораторная работа №9. Реализация сервера РВС по технологии RMI.....	113
3.3 Лабораторная работа №10. Реализация клиента РВС по технологии RMI.....	113
4 WEB-ТЕХНОЛОГИИ РАСПРЕДЕЛЁННЫХ СИСТЕМ.....	114
4.1 Лабораторная работа №11. Технология сервлетов на базе сервера Apache Tomcat.....	114
4.2 Лабораторная работа №12. Технология JSP для формирования динамических HTML-страниц.....	114
4.3 Лабораторная работа №13. Шаблон проектирования MVC.....	114
5 СЕРВИС-ОРИЕНТИРОВАННЫЕ АРХИТЕКТУРЫ.....	115
5.1 Лабораторная работа №14. Проектирование трёхзвенной РВС в стиле REST.....	115
5.2 Лабораторная работа №15. Реализация модели приложения в стиле REST.....	115
5.3 Лабораторная работа №16. Реализация распределённого приложения в стиле REST...	115
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	116

ВВЕДЕНИЕ

Методические указания содержат учебный материал для проведения лабораторных работ по дисциплине «Распределенные вычислительные системы» (РВС), что предполагает базовый уровень подготовки бакалавриата. Теоретический материал этой дисциплины изложен в учебном пособии [1]. Он также содержит поясняющие примеры на языке Java положенные в основу данных методических указаний. Дополнительно может быть использовано учебно-методическое пособие [2], которое ранее использовалось для проведения практических занятий по изучаемой дисциплине. При этом предполагается, что студент ранее не изучал язык Java, но прошёл успешное обучение по изучению языков C/C++. Для полноценного изучения языка Java студенту рекомендуется обращаться к источнику [3].

Цель данного пособия — базовое организационное и методическое обеспечение процессов выполнения лабораторных работ по дисциплине РВС в объёме предусмотренном учебными планами вуза.

Основная задача данных указаний — практическое закрепление навыков программирования на языке Java, достаточных для реализации элементов различных распределённых систем.

Оглавление данных методических указаний привязано к разделам базового учебного пособия [1] по дисциплине РВС, что соответствует следующему списку:

1. Введение в теорию вычислительных сетей.
2. Инструментальные средства языка Java.
3. Объектные распределенные системы.
4. Web-технологии распределенных систем.
5. Сервис-ориентированные архитектуры.

Первый раздел указаний посвящен двум лабораторным работам объединённым под общим названием «*Тестирование системного и инструментального ПО рабочей области студента*». Работа №1 ориентирована на подготовку компьютерной обучающей среды для выполнения последующих лабораторных работ. Она опирается на специальный программный комплекс ОС УПК АСУ, который кратко описан в отдельном методическом пособии [4]. Работа №2 дополняет работу №1 в плане изучения инструментальных средств Eclipse EE, СУБД Derby, сервера приложений TomEE.

Второй раздел полностью посвящён изучению языка Java и содержит описание пяти лабораторных работ. Учебная цель этих работ — изучение синтаксиса и семантики языка Java, а также получение навыков программирования базовых прикладных задач, включающих сетевые задачи и работу с базами данных.

Остальные три раздела методических указаний содержат девять лабораторных работ посвящённых различным технологиям реализации распределённых систем. Тематика этих работ соответствует тематике заявленных выше трёх последних тем.

Результаты всех лабораторных работ оформляются общим отчётом, шаблон которого предоставляется студенту в его индивидуальной рабочей области.

1 ВВЕДЕНИЕ В ТЕОРИЮ ВЫЧИСЛИТЕЛЬНЫХ СЕТЕЙ

Данный раздел методических указаний посвящен двум лабораторным работам, которые объединены под общим названием *«Тестирование системного и инструментального ПО рабочей области студента»*. Формально, в теоретическом плане, он относится к первому разделу учебного пособия [1], но практически связан с изучением программной инфраструктуры, которая касается учебной среды выполнения всех лабораторных работ.

Работа №1 *«Тестирование ПО рабочей области студента»* ориентирована на подготовку компьютерной обучающей среды для выполнения всех последующих лабораторных работ. Она опирается на специальный программный комплекс ОС УПК АСУ, который кратко описан в отдельном методическом пособии [4].

Работа №2 *«Инструментальные средства реализации распределённых систем»* дополняет работу №1 в плане общего архитектурного изучения инструментальных средств Eclipse EE, СУБД Apache Derby и сервера приложений TomEE.

Примечание — Все используемые в методических указаниях внешние источники информации написаны в разное время и могут содержать разные обозначения или ссылаться на программное обеспечение различных версий, поэтому при выполнении лабораторных работ студент должен ориентироваться на данные методические указания. В случае непонятных студенту противоречий следует обратиться за разъяснениями к преподавателю, ведущему лабораторные занятия.

Общие правила выполнения лабораторных работ соответствуют общим правилам, которые студент освоил при изучении дисциплины *«Операционные системы»*:

- 1) лабораторные работы выполняются в среде ОС УПК АСУ и должны быть описаны в едином отчёте, шаблон которого имеется в каждой индивидуальной рабочей области, размещённой на личном flashUSB студента;
- 2) студент персонально отвечает как за сохранность своей рабочей области, так и за сохранность своего отчета;
- 3) шаблон отчёта содержит необходимые оглавления для всех лабораторных работ;
- 4) студент может по своему усмотрению добавлять различные пункты описания лабораторных работ;
- 5) в качестве общих правил оформления личного отчёта студенту следует ориентироваться на правила оформления данных методических указаний.

Примечание — Общая временная нагрузка на выполнение первых двух работ является различной, поэтому после выполнения первой работы необходимо приступить к выполнению второй работы.

1.1 Лабораторная работа №1.

Тестирование ПО рабочей области студента

Учебная цель данной работы — подготовка и изучение инструментальных программных средств, необходимых для успешного выполнения учебных заданий по изучаемой дисциплине.

Результатом выполнения данной лабораторной работы должны быть:

- 1) знания о структуре и местоположении компонент дистрибутива ОС УПК АСУ, включающего ПО используемых инструментальных средств и ПО личной рабочей области студента;
- 2) знания о составе и местоположении учебного материала изучаемой дисциплины;
- 3) текст отчёта, записанный в файле *ОтчетРВС.odt*.

1.1.1 Общая структура ПО для проведения лабораторных работ

Общее описание дистрибутива ОС УПК АСУ опубликовано в учебно-методическом пособии [4], размещённом в электронном виде на портале ТУСУР. Это пособие студент уже изучал в дисциплине «Операционные системы».

Примечание - Все дополнения, касающиеся непосредственного выполнения лабораторных работ, размещения студентов по рабочим местам компьютерных классов и последовательности выполнения учебных заданий производятся преподавателем во время проведения конкретных учебных занятий.

Полный комплект дистрибутива ОС УПК АСУ размещается в отдельном каталоге **C:\asu64upk** корня файловой системы MS Windows. Такое размещение подробно описано в источнике [4, подраздел 1.1] и полностью соответствует используемой архитектуре учебного ПО, установленного в учебных классах кафедры АСУ.

Студент должен убедиться в наличии на диске **C:** каталога **asu64upk**, а также проверить его структуру согласно данным таблицы 1.1.

Таблица 1.1 — Назначение компонент дистрибутива ОС УПК АСУ

Компонента	Назначение компоненты
boot	Каталог размещения ядра ОС и временной файловой системы.
opt	Каталог размещения дополнительного инструментального ПО (см. таблицу 1.2).
themes	Каталог размещения рабочей области студента после подключения его личного архива.
upkasu	Каталог размещения ПО ОС УПК АСУ.
upk_asu.pdf	Файл учебно-методического пособия, который следует использовать вместо источника [4].

Следует убедиться, что в каталоге **C:\asu64upk\opt** находятся файлы дополнительного программного обеспечения, представленные в таблице 1.2.

Таблица 1.2 — Файлы дополнительного программного обеспечения ОС УПК АСУ

Файл	Назначение файла
<i>apache-tomee-plume-803.sfs</i>	Дистрибутив сервера приложений Apache TomEE, включающий сервер Apache Tomcat.
<i>db-derby-10.14.sfs</i>	Дистрибутив СУБД Apache Derby.
<i>eclipse-jee-2020.sfs</i>	Дистрибутив инструментальной среды разработки приложений EclipseEE.

Первое учебное задание:

- 1) загрузить на компьютере ОС MS Windows и проверить структуру дистрибутива ОС УПК АСУ и наличие файлов согласно содержанию таблиц 1.1 и 1.2;
- 2) взять у преподавателя файл *rvs-home.ext4fs.gz* и поместить его на личный flashUSB в каталог *X:\asu64upk\themes*;
- 3) выполнить загрузку ОС УПК АСУ с личного flashUSB, подключиться пользователем *asu* и подключить личную flashUSB с помощью значка запуска на рабочем столе;
- 4) выполнить стандартное подключение темы обучения с именем *rvs*, отмонтировать личную flashUSB и выйти из сеанса пользователя *asu*;
- 5) подключиться к сеансу пользователя *upk*, после чего перейти к выполнению следующего пункта лабораторной работы.

Примечание — Все общие правила работы с ОС УПК АСУ и личной рабочей областью изучаемой дисциплины полностью соответствуют правилам полученным студентом при изучении дисциплины «Операционные системы».

1.1.2 Рабочий стол пользователя *upk* и инструменты его рабочей области

Согласно общей традиции *рабочий стол* пользователя *upk* для дисциплины «Распределенные вычислительные системы» имеет свою заставку, показанную ниже на рисунке 1.1.

Если заставка рабочего стола не соответствует рисунку 1.1, то её нужно установить самостоятельно: нужный файл находится в каталоге */home/upk/Изображения*.

На рабочем столе пользователя *upk* находятся следующие *значки*:

- 1) *Учебный материал* — ссылка на каталог с файлами учебных материалов по изучаемой дисциплине;
- 2) *upk_asu.pdf* — ссылка на обновленный документ ОС УПК АСУ;
- 3) *ОтчетPBC.odt* — ссылка на шаблон личного отчёта студента;
- 4) *EclipseEE* — значок запуска среды разработки Eclipse EE, назначение и использование которого будет изучаться в следующей лабораторной работе.

Второе учебное задание:

- 1) проверить наличие указанных выше компонент рабочего стола пользователя *upk*, а также - присутствие необходимых учебно-методических пособий (УМП);
- 2) запустить на редактирование файл отчёта и зафиксировать в нём выполнение заданий по пунктам 1.1.1 и 1.1.2.



Рисунок 1.1 — Изображение рабочего стола пользователя upk

1.1.3 Назначение каталогов рабочей области студента

Учебные задания изучаемой дисциплины используют дополнительное программное обеспечение (ПО), которое необходимо подключать в начале каждого занятия по лабораторным работам. Такое ПО подключается к фиксированным каталогам, назначение и список которых студент должен хорошо изучить.

Подключаемые каталоги инструментальных средств — каталоги для монтирования программного обеспечения (ПО) инструментальных средств, необходимых для выполнения лабораторных работ. Их список представлен в таблице 1.3.

Таблица 1.3 — Каталоги монтирования ПО инструментальных средств ОС УПК АСУ

Каталог	Назначение каталога
<i>/opt/tomee</i>	Точка монтирования файла <i>apache-tomee-plume-803.sfs</i> дистрибутива сервера приложений Apache TomEE, включающего сервер Apache Tomcat.
<i>/opt/derby</i>	Точка монтирования файла <i>db-derby-10.14.sfs</i> дистрибутива СУБД Apache Derby.
<i>/opt/eclipseEE</i>	Точка монтирования файла <i>eclipse-jee-2020.sfs</i> дистрибутив инструментальной среды разработки приложений EclipseEE.

Рабочие каталоги инструментальных средств — каталоги рабочей области пользователя *upk*, используемые для локального отображения ПО инструментальных средств ОС УПК АСУ, а также для размещения результатов выполнения лабораторных работ. Их список представлен в таблице 1.4.

Таблица 1.4 — Рабочие каталоги ПО инструментальных средств пользователя *upk*

Каталог	Назначение каталога
<i>~/tomee</i>	Каталог для локального размещения файлов каталога <i>/opt/tomee</i> , кроме каталога библиотек. Используется для локальных настроек сервера Apache TomEE для пользователя <i>upk</i> .
<i>~/databases</i>	Каталог для размещения специальных сценариев и баз данных дистрибутива СУБД Apache Derby.
<i>~/rvs</i>	Каталог для размещения создаваемых студентом проектов в инструментальной среде разработки Eclipse EE. Первоначально, если каталог не пустой, то необходимо его создать заново, чтобы не создавать проблем для учебных проектов.
<i>~/src/rvs</i>	Каталог для размещения различных сценариев и данных, которые потребуются в дальнейших лабораторных работах.
<i>~/lib</i>	Каталог для размещения результатов выполняемых проектов. Содержимое каталога поясняется при выполнении последующих лабораторных работ.

Третье учебное задание:

- 1) проверить содержимое каталогов таблицы 1.3: они должны быть пустыми после авторизации пользователем *upk*;
- 2) убедиться в наличии всех каталогов таблицы 1.4 в рабочей области пользователя *upk*; их содержимое поясняется по мере выполнения последующих заданий;
- 3) зафиксировать в личном отчёте наиболее важные сведения, полученные при выполнении этого задания.

1.1.4 Подключение инструментальных средств к рабочей среде студента

Задача данного пункта работы — завершение базового тестирования учебной программной среды пользователя *upk*, обеспечивающего подключение (монтирования) инструментального ПО для выполнения лабораторных работ по изучаемой дисциплине.

Исходные данные для выполнения работы:

- 1) после запуска ОС УПК АСУ том *C:* с компонентами используемого дистрибутива автоматически монтируется к каталогу */run/basefs* и становится доступным пользователю *upk*;
- 2) в результате пользователю *upk* становятся доступными для монтирования файлы дополнительного программного обеспечения перечисленные ранее в таблице 1.1;
- 3) в каталоге *~/bin* пользователя *upk* находятся файлы сценариев предназначенные для монтирования инструментальных средств к каталогам рабочей области студента перечисленные ранее в таблице 1.3;
- 4) список используемых для монтирования сценариев представлен на рисунке 1.2:
- 5) *mountDerby* — монтирует СУБД Apache Derby к каталогу */opt/derby*;
- 6) *mountEclipseEE* — монтирует EclipseEE к каталогу */opt/eclipseEE*;
- 7) *mountTomee* — монтирует сервер приложений к каталогу */opt/tomee*;
- 8) демонтаж любого из указанных выше инструментальных средств осуществляется в командной строке терминала, согласно выражения (1.1).

sudo umount /opt/имя_каталога

(1.1)

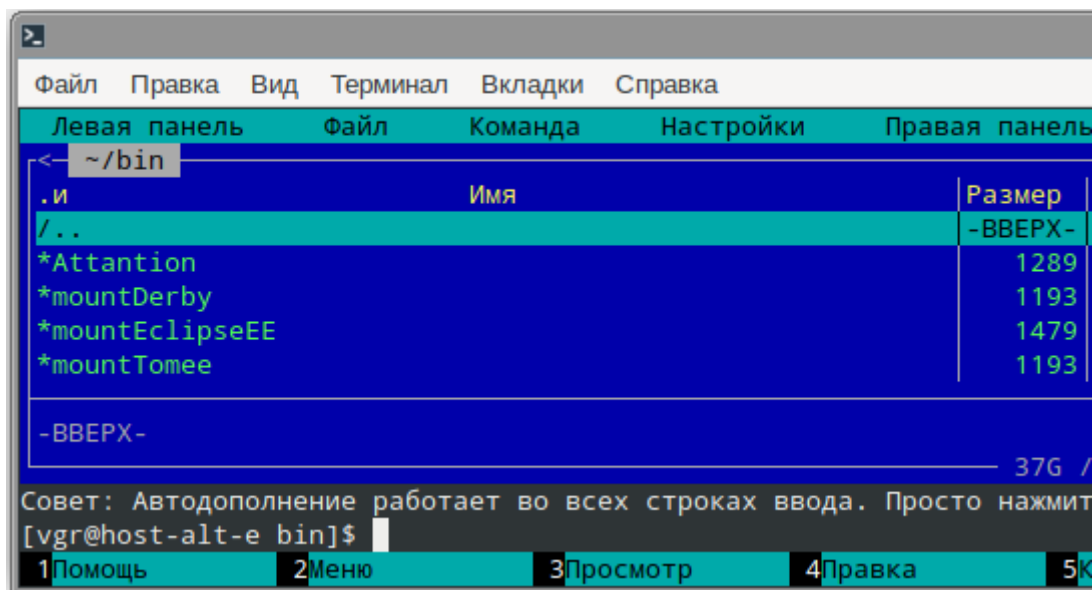


Рисунок 1.2 — Сценарии каталога ~/bin

Четвёртое учебное задание:

- 1) запустить Midnight Commander, перейти в каталог *~/bin* и последовательно запустить сценарии монтирования инструментальных средств СУБД Derby, сервера приложений TomEE и среды разработки Eclipse EE;
- 2) перейти в каталог */opt* и проверить результаты монтирования всех инструментальных средств;
- 3) с помощью шаблона выражения (1.1) провести «ручное» размонтирование инструментальных средств;
- 4) описать результаты работы по этому пункту в личном отчёте;
- 5) выбрать один из показанных на рисунке 1.2 сценариев монтирования и описать его работу в личном отчёте;
- 6) сохранить отчёт на личном flashUSB студента;
- 7) провести штатное завершение работы с сохранением рабочей области пользователя *upk* на личном flashUSB.

Примечание — После завершения данной лабораторной работы необходимо снова запустить ОС УПК АСУ и приступить к выполнению лабораторной работы №2.

1.2 Лабораторная работа №2. Инструментальные средства реализации распределённых систем

Данная лабораторная работа является технологическим продолжением лабораторной работы №1. Она предназначена для завершающей настройки рабочей области студента, необходимой для подготовки и проверки работоспособности её инструментальных средств, обеспечивающих практическое освоение учебного материала изучаемой дисциплины.

Общая задача данной работы — тестирование и настройка базовых инструментальных средств достаточных для проектирования и реализации широкого класса распределённых систем (РВС).

Познавательная задача работы — первое знакомство с инструментальными средствами разработки систем РВС, основанными на языке объектно-ориентированного программирования Java. К таким средствам относятся: инструментальные средства самого языка Java, организованного на уровнях Standard и Enterprise Edition, а также средства инструментальных средств СУБД и серверов приложений.

С учётом общей и познавательной задач, общее выполнение лабораторной работы разделено на четыре последовательных этапа:

- 1) тестирование ПО языка Java;
- 2) тестирование ПО СУБД Apache Derby;
- 3) тестирование ПО сервера приложений Apache TomEE;
- 4) тестирование ПО среды разработки Eclipse EE.

Таким образом по результатам выполнения второй лабораторной работы студент должен получить первые впечатления о целевом («джентельменском») наборе инструментальных средств, которым должен владеть любой специалист информационных технологий (ИТ) ориентированный на разработку распределённых систем.

1.2.1 Тестирование ПО языка Java

Объектно ориентированный язык Java широко используется в технологиях создания распределённых вычислительных систем. В изучаемой дисциплине он является базовым инструментальным программным средством, на основе которого проводятся все лабораторные занятия.

Учебная задача данного пункта — тестирование работоспособности базовых инструментальных средств языка Java, обеспечивающих как создание, так и запуск программ, написанных на языке Java в среде ОС Linux.

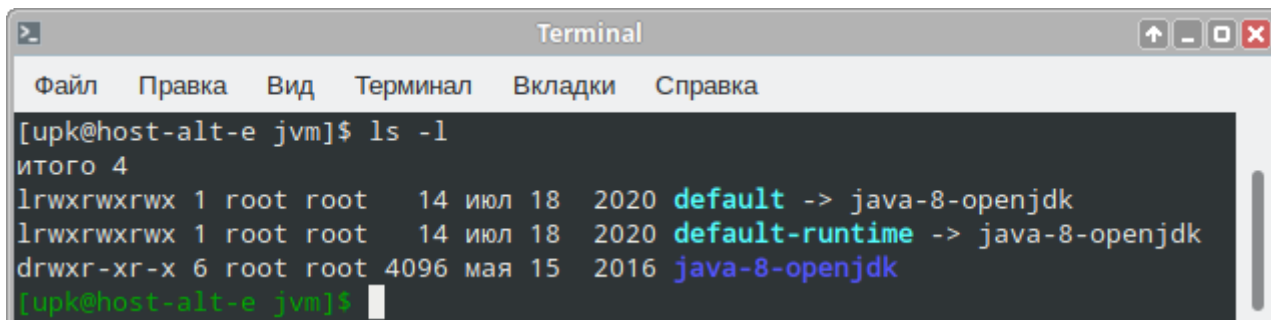
Решение поставленной задачи опирается на выполнение следующих двух операций:

- 1) определение местоположения и состава дистрибутивов ПО языка Java;
- 2) настройка и тестирование операционной среды ОС Linux, обеспечивающей нормальный запуск, по крайней мере, двух команд: *java* и *javac*.

Определение местоположения дистрибутивов языка Java.

В ОС Linux стандартное размещение дистрибутивов языка Java определено в каталоге */usr/lib/jvm*.

На рисунке 1.3 показано содержимое каталога размещения дистрибутивов Java для текущей версии ОС УПК АСУ.



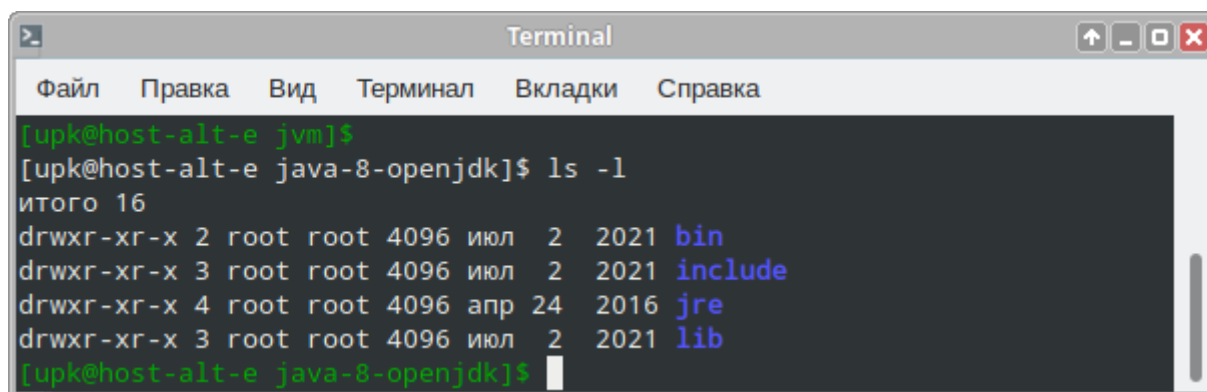
```
Terminal
Файл  Правка  Вид  Терминал  Вкладки  Справка
[upk@host-alt-e jvm]$ ls -l
итого 4
lrwxrwxrwx 1 root root 14 июл 18 2020 default -> java-8-openjdk
lrwxrwxrwx 1 root root 14 июл 18 2020 default-runtime -> java-8-openjdk
drwxr-xr-x 6 root root 4096 мая 15 2016 java-8-openjdk
[upk@host-alt-e jvm]$
```

Рисунок 1.3 — Содержимое каталога /usr/lib/jvm

Хорошо видно, что:

- 1) в системе установлен единственный дистрибутив **java-8-openjdk**, размещённый в одноименном каталоге;
- 2) на указанный каталог имеются две символичные ссылки: **default** и **default-runtime**.

Если перейти в каталог **usr/lib/jvm/java-8-openjdk**, то мы увидим типичную для ОС UNIX структуру размещения программного обеспечения, показанную на рисунке 1.4.



```
Terminal
Файл  Правка  Вид  Терминал  Вкладки  Справка
[upk@host-alt-e jvm]$
[upk@host-alt-e java-8-openjdk]$ ls -l
итого 16
drwxr-xr-x 2 root root 4096 июл 2 2021 bin
drwxr-xr-x 3 root root 4096 июл 2 2021 include
drwxr-xr-x 4 root root 4096 апр 24 2016 jre
drwxr-xr-x 3 root root 4096 июл 2 2021 lib
[upk@host-alt-e java-8-openjdk]$
```

Рисунок 1.4 — Содержимое каталога /usr/lib/jvm/java-8-openjdk

Назначение перечисленных каталогов следующее:

- 1) **bin** — каталог размещения различных исполняемых файлов среды разработки JDK (Java Development Kit), содержащих такие программы как: **java**, **javac**, **jar** и другие;
- 2) **include** — каталог размещения файлов описателей языка C, которые нам неинтересны и не используются;
- 3) **jre** — каталог размещения дистрибутива JRE (Java Runtime Environment) — среды исполнения языка Java, той же версии, что и версия JDK;
- 4) **lib** — каталог размещения файлов библиотек JDK.

Соответственно, если далее перейти в каталог **jre**, то мы увидим аналогичный список каталогов, показанных на рисунке 1.5, но относящихся к среде Java Runtime Environment и обеспечивающие набор исполняемых файлов и библиотек для работы ПО Java.

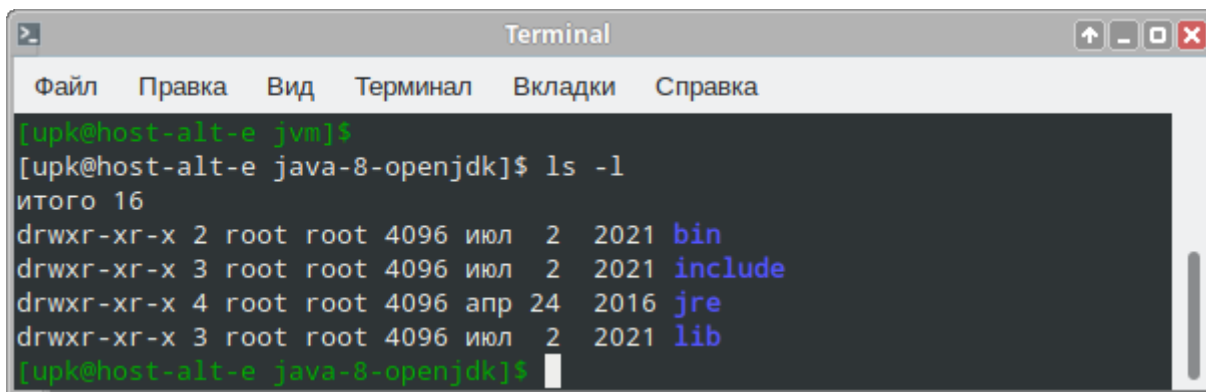


Рисунок 1.5 — Содержимое каталога /usr/lib/jvm/java-8-openjdk/jre

Примечание — В различных дистрибутивах ОС Linux, каталог */usr/lib/jvm* может содержать множество различных дистрибутивов JDK или отдельных JRE, поэтому студент должен хорошо изучить уже установленное или необходимое ПО Java.

Первое учебное задание:

- 1) запустить Midnight Commander в окне виртуального терминала;
- 2) перейти в каталог */usr/lib/jvm* и изучить содержимое установленных в ОС дистрибутивов языка Java;
- 3) наиболее важные сведения отобразить в личном отчёте.

Настройка и тестирование операционной среды ОС Linux.

Индивидуальные настройки операционной среды ОС Linux осуществляются заданием *глобальных переменных среды пользователя* записываемых в файле *\$HOME/.bashrc*, первоначальное содержимое которого показано на рисунке 1.6.

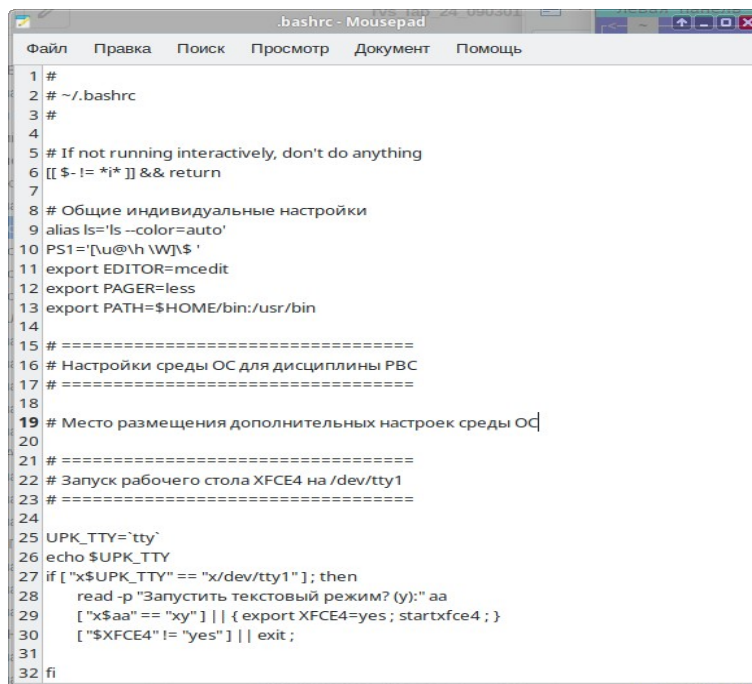


Рисунок 1.6 — Первоначальное содержимое файла \$HOME/.bashrc

При запуске виртуального терминала файл **\$HOME/.bashrc**, представляющий собой сценарий языка **bash**, выполняется автоматически. Одновременно настраиваются индивидуальные переменные среды доступные в среде запущенного терминала.

Для настройки и тестирования базового ПО JDK и JRE языка Java необходимо правильно задать и сделать глобальными для пользователя **upk** следующие переменные среды ОС:

- 1) **JAVA_HOME** — указывает на каталог размещения дистрибутива JDK;
- 2) **JAVA_JRE** — указывает на каталог размещения дистрибутива JRE;
- 3) **CLASSPATH** — указывает на каталоги и файлы дополнительных библиотек языка Java, размещённых вне дистрибутивов JDK и JRE;
- 4) **PATH** — общий список каталогов просматриваемых в среде ОС во время запуска исполняемых файлов; в нашем случае должен содержать каталоги **bin** дистрибутивов JDK и JRE.

Правильные значения перечисленных переменных следует записать в файл **.bashrc**, в место помеченное на рисунке 1.6 как «*# Место размещения дополнительных настроек ОС*».

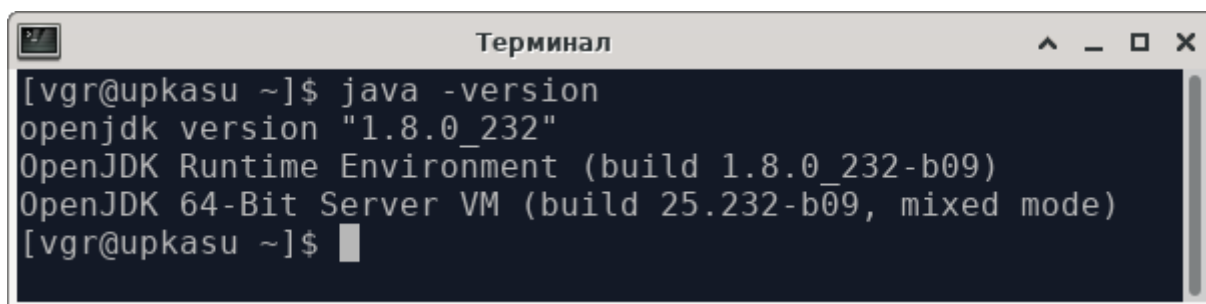
Для дистрибутива ОС УПК АСУ указанные выше настройки будут иметь вид, показанный на рисунке 1.7.

Рисунок 1.7 — Базовые настройки среды Java добавленные в файл **~/.bashrc**

Правильность базовых настроек среды языка Java проверяется после сохранения файла **~/.bashrc** в новом запущенном виртуальном терминале. Для проверки правильности произведённых настроек среды пользователя **upk** достаточно использования трёх команд, заданных выражениями (1.2) — (1.4).

java -version	(1.2)
javac -version	(1.3)
echo \$имя_переменной_среды	(1.4)

Например, на рисунке 1.8 показан пример проверки запуска виртуальной машины Java, выводящей версию дистрибутивов JRE, JDK и другие стандартные сообщения.



```
Терминал
[vgr@upkasu ~]$ java -version
openjdk version "1.8.0_232"
OpenJDK Runtime Environment (build 1.8.0_232-b09)
OpenJDK 64-Bit Server VM (build 25.232-b09, mixed mode)
[vgr@upkasu ~]$
```

Рисунок 1.8 — Проверка запуска виртуальной машины языка Java

Второе учебное задание:

- 1) отредактировать файл `~/.bashrc` в соответствии с данными первого учебного задания данной лабораторной работы;
- 2) запустить виртуальный терминал и выполнить проверку произведённых настроек переменных среды пользователя *upk*;
- 3) результаты работы отразить в личном отчёте.

1.2.2 Тестирование ПО СУБД Apache Derby

Проект Apache Derby, появившийся в 1997 году, является примером СУБД полностью написанным на языке Java. Его характеризуют достаточно малые размеры, высокое быстродействие и возможность работы как в сетевом варианте (режим *networkserver*), так и во встроённом режиме (режим *embedded*).

Примечание — Подробное изучение настроек СУБД будет проведено в конце второй темы учебного пособия [1] и закреплено при выполнении лабораторной работы №7. В данном пункте рассматриваются только вопросы выбора приемлемой версии дистрибутива Apache Derby, которая должна быть согласована с используемой версией платформы языка Java.

Поддерживаемые версии дистрибутивов СУБД Apache Derby размещены на его официальном сайте <https://db.apache.org/derby/>.

Перейдя на страницу Download, показанную на рисунке 1.9, мы увидим, что нашей версии платформы Java 8 соответствует последняя версия 10.14.2.0 искомой СУБД. Далее, перейдя по ссылке выбранного дистрибутива, можно скачать файл *db-derby-10.14.2.0-bin.tar.gz*, соответствующий нужной версии СУБД.

Примечание — Нужная версия и формат дистрибутива Apache Derby уже установлена в среде ОС УПК АСУ и была рассмотрена в первой лабораторной работе. Приведённые ниже рассуждения лишь показывают необходимые действия, если бы студент самостоятельно формировал нужный формат.

Скачанный с сайта файл следует разместить в отдельной пустой директории установки и распаковывать командой (1.5):

```
sudo tar -xvfz db-derby-10.14.2.0-bin.tar.gz (1.5)
```

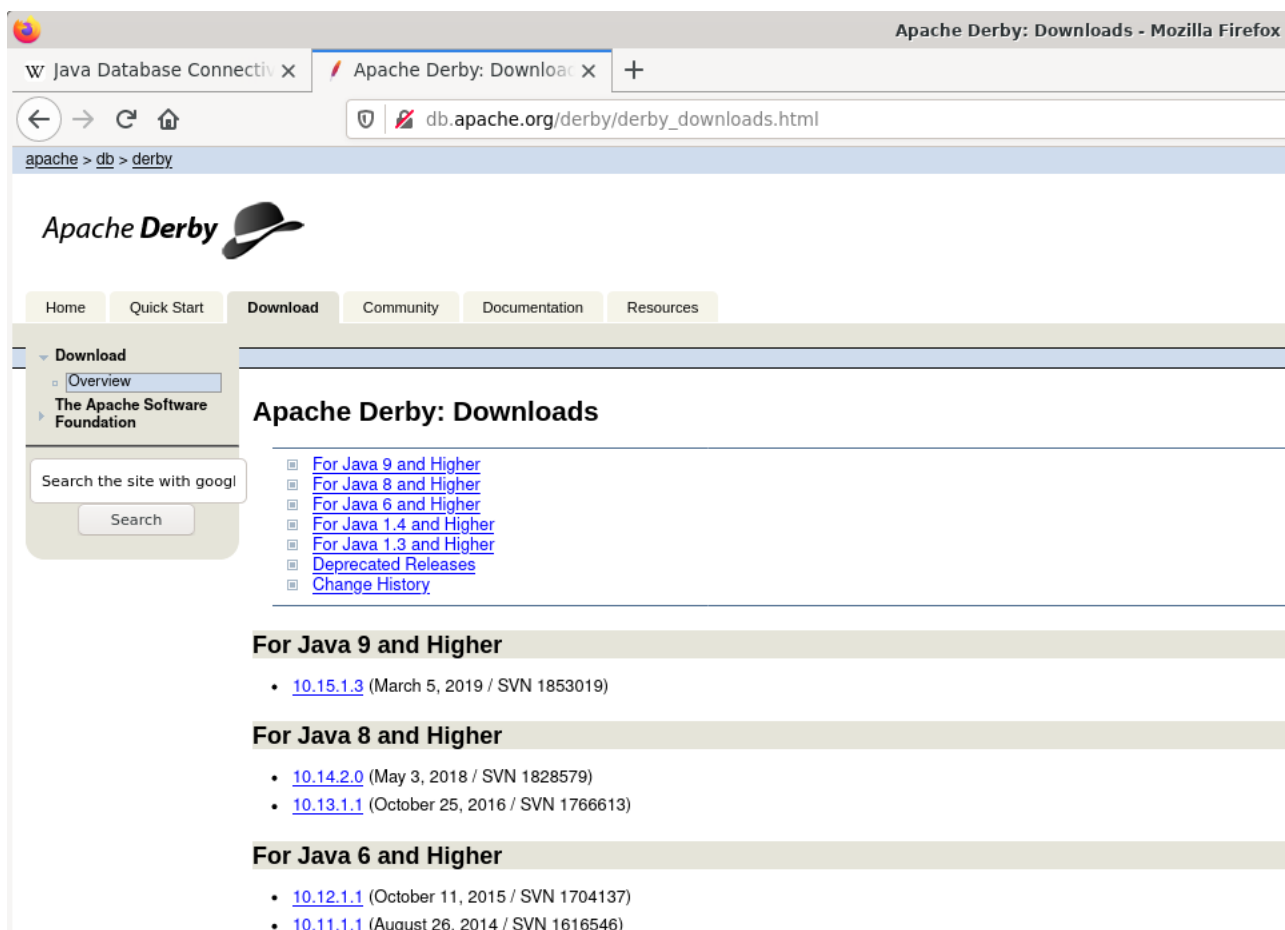



Рисунок 1.9 — Страница выбора дистрибутивов СУБД на сайте Apache Derby

В результате выполнения действия выражения (1.5) создаётся каталог со стандартным именем ***db-derby-10.14.2.0-bin***, который может рассматриваться как корневой каталог СУБД Apache Derby.

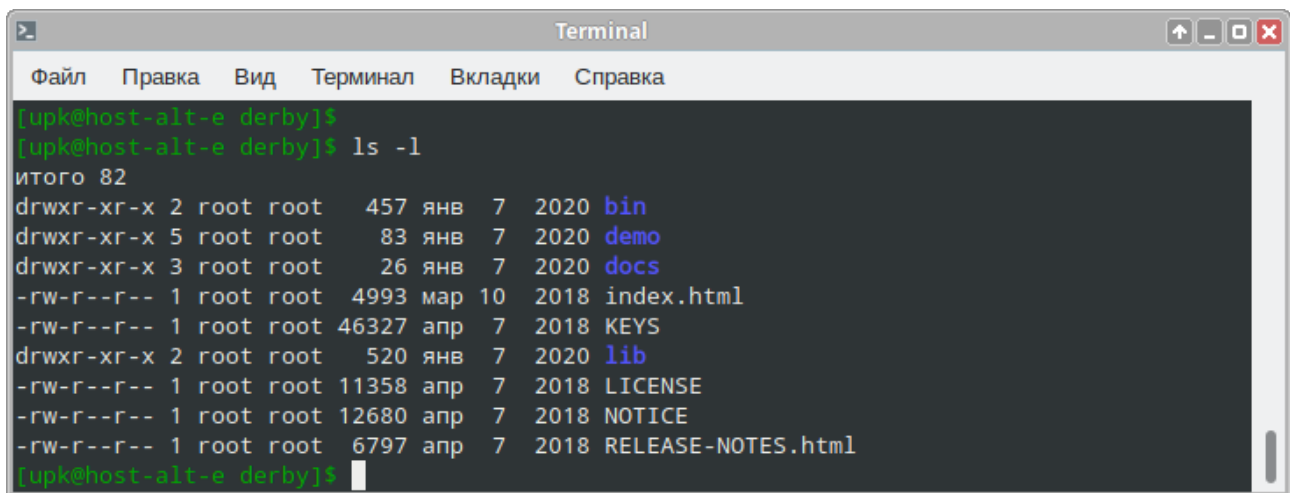
Примечание — По причине ограниченных ресурсных возможностей ОС УПК АСУ из полученного дистрибутива удалены излишняя документация и примеры, а сам каталог сжат утилитой ***mksquashfs***. Результат таких преобразований в виде файла ***db-derby-10.14.sfs*** используется в лабораторных работах изучаемой дисциплины.

Первоначальное тестирование ПО СУБД Apache Derby разбивается на два этапа: *настройка среды пользователя ОС УПК АСУ* и *тестирование работоспособности ПО СУБД Apache Derby*.

Настройка среды пользователя ОС УПК АСУ.

Воспользуемся знаниями выполнения первой лабораторной работы, которые получены при выполнении [пункта 1.1.4](#):

- 1) сначала выполним сценарий ***~/bin/mountDerby***, который монтирует дистрибутив ПО Apache Derby к каталогу ***/opt/derby***;
- 2) затем перейдём в каталог ***/opt/derby*** и выполним команду ***ls -l***, которая выведет на терминал структуру ПО Apache Derby, как это показано на рисунке 1.10.



```
[upk@host-alt-e derby]$  
[upk@host-alt-e derby]$ ls -l  
итого 82  
drwxr-xr-x 2 root root 457 янв 7 2020 bin  
drwxr-xr-x 5 root root 83 янв 7 2020 demo  
drwxr-xr-x 3 root root 26 янв 7 2020 docs  
-rw-r--r-- 1 root root 4993 мар 10 2018 index.html  
-rw-r--r-- 1 root root 46327 апр 7 2018 KEYS  
drwxr-xr-x 2 root root 520 янв 7 2020 lib  
-rw-r--r-- 1 root root 11358 апр 7 2018 LICENSE  
-rw-r--r-- 1 root root 12680 апр 7 2018 NOTICE  
-rw-r--r-- 1 root root 6797 апр 7 2018 RELEASE-NOTES.html  
[upk@host-alt-e derby]$
```

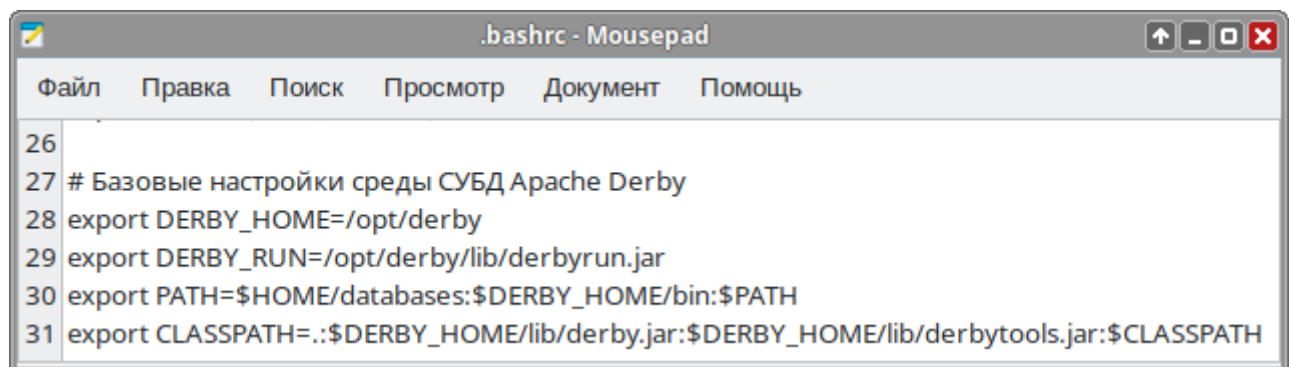
Рисунок 1.10 — Содержимое каталога /opt/derby

Общий анализ рисунка 1.10 показывает, что структура ПО Apache Derby соответствует общей структуре дистрибутивов ПО ОС UNIX и Linux:

- 1) каталог **bin** содержит запускаемые на исполнение файлы; в основном это — сценарии на языке Bourne shell самостоятельно настраивающие среду ОС пользователя, которая необходима для правильного функционирования ПО;
- 2) каталог **lib** содержит библиотеки в формате файлов ***.jar**, необходимые для функционирования ПО Apache Derby.

Главным ключевым параметром, обеспечивающим правильное функционирование ПО Apache Derby, является глобальная переменная **DERBY_HOME**, значение которой должно указывать на *вершину каталога* установленного ПО дистрибутива.

Таким образом для настройки среды пользователя ОС УПК АСУ, обеспечивающей базовое функционирование ПО Apache Derby, необходимо в файл **~/.bashrc** следует добавить записи представленные на рисунке 1.11.



```
.bashrc - Mousepad  
Файл Правка Поиск Просмотр Документ Помощь  
26  
27 # Базовые настройки среды СУБД Apache Derby  
28 export DERBY_HOME=/opt/derby  
29 export DERBY_RUN=/opt/derby/lib/derbyrun.jar  
30 export PATH=$HOME/databases:$DERBY_HOME/bin:$PATH  
31 export CLASSPATH=.:$DERBY_HOME/lib/derby.jar:$DERBY_HOME/lib/derbytools.jar:$CLASSPATH
```

Рисунок 1.11 — Базовые настройки среды Apache Derby добавленные в файл ~/.bashrc

Примечание — Настройки среды пользователя показанные на рисунке 1.11 следует добавить после настроек языка Java, показанных ранее на рисунке 1.7.

Следует также правильно понимать семантику настроек, показанных на рисунке 1.7:

- 1) запись параметра **DERBY_HOME** задаёт начало координат дистрибутива Apache Derby в файловой системе ОС;

- 2) запись параметра **DERBY_RUN** является необязательной; она в дальнейшем используется в различных сценариях, обеспечивающих сетевой запуск сервера Apache Derby;
- 3) запись параметра **PATH** рекуррентно добавляет каталоги, в которых размещаются запускаемые программы и сценарии.

Тестирование работоспособности ПО СУБД Apache Derby.

Первоначальное тестирование работоспособности ПО Apache Derby достаточно провести на примере запуска утилиты **ij**.

ij (Interactive JDBC) — стандартная утилита ПО Apache Derby, размещённая в каталоге **\$DERBY_HOME/bin** и предназначенная как для интерактивного доступа к базам данных, так и для исполнения сценариев на языке SQL.

На рисунке 1.12 показан тестовый запуск утилиты **ij** в окне виртуального терминала:

- 1) без параметров утилита **ij** запускается в интерактивном режиме и выводит на терминал свою версию, а затем — приглашение для ввода команд;
- 2) ввод команд **help** и **exit** сначала выводит текст со списком команд, а затем — завершает работу утилиты.

```
[uprk@host-alt-e Рабочий стол]$ ij
версия ij 10.14
ij> help;exit;

Поддерживаемые команды:

PROTOCOL 'протокол JDBC' [ AS идент ];
    -- задать протокол по умолчанию или указанный протокол
DRIVER 'класс драйвера';    -- загрузить указанный класс
CONNECT 'url базы данных' [ PROTOCOL указанный_протокол ] [ AS имя_соединения ];
    -- соединиться с указанным URL базы данных
    -- и также позволяет назначить идентификатор
SET CONNECTION имя_соединения; -- переключиться на указанное соединение
SHOW CONNECTIONS;              -- вызвать список всех соединений
AUTOCOMMIT [ ON | OFF ];      -- задать режим автоматического принятия для соединения
DISCONNECT [ CURRENT | имя_соединения | ALL ];
    -- удалить текущее или указанное соединение либо все соединения;
    -- значение по умолчанию - CURRENT (текущее соединение)

SHOW SCHEMAS;                  -- вызвать список всех схем в текущей базе данных
SHOW [ TABLES | VIEWS | PROCEDURES | SYNONYMS ] { IN схема };
    -- вызвать список таблиц, просмотров, процедур или синонимов
SHOW INDEXES { IN схема | FROM таблица };
```

Рисунок 1.12 — Проверка настроек ПО Apache Derby с помощью утилиты **ij**

Примечание — Одно из преимуществ СУБД Apache Derby — способность функционировать во встроенном режиме (режим **embedded**). Это позволяет любой программе на языке Java легко создавать и использовать эту СУБД. Проверка настроек ПО с помощью утилиты **ij** позволяет, что локальный (встроенный) режим работает нормально.

Для проверки работоспособности ПО Apache Derby в сетевом режиме следует перейти в каталог **~/databases**, как это показано на рисунке 1.13, и изучить исходный текст сценариев **startServerDB** и **stopServer**:

- 1) **startServerDB** — сценарий на языке Bourne shell, запускающий ПО СУБД Apache Derby в сетевом режиме;

- 2) **stopServer** — сценарий на языке Bourne shell, останавливающий сетевой режим ПО СУБД Apache Derby.

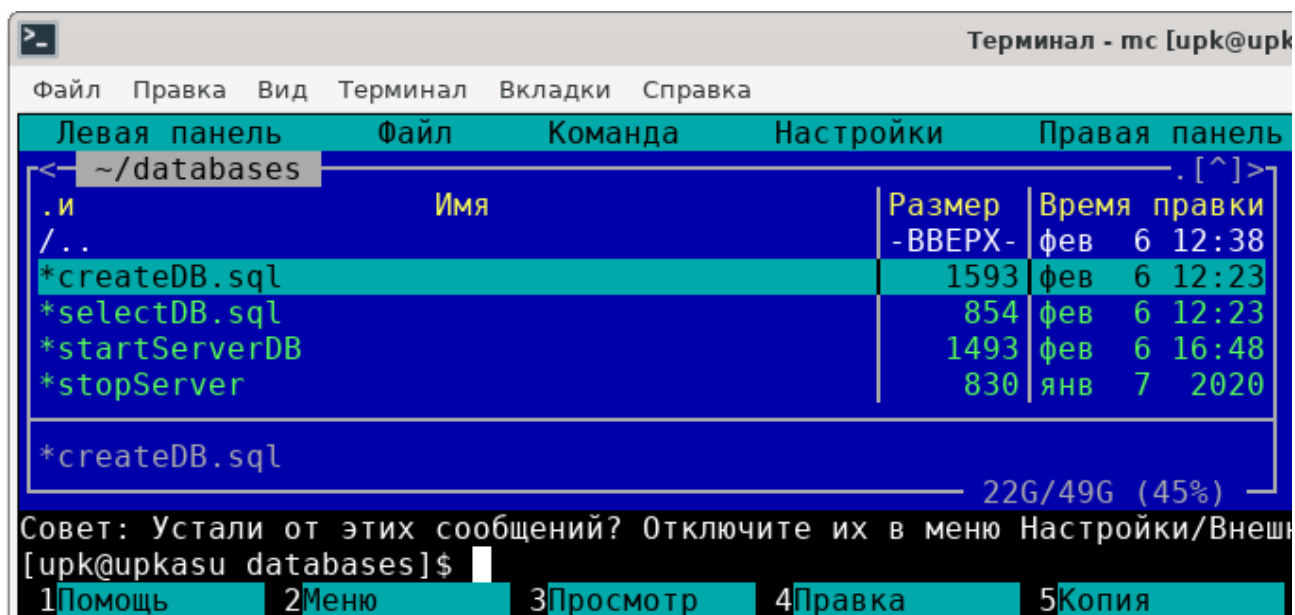


Рисунок 1.13 — Сценарии каталога размещения сценариев и баз данных СУБД Apache Derby

Третье учебное задание:

- 1) подключить к системе дистрибутив ПО Apache Derby, перейти в каталог `/opt/derby` и исследовать содержимое каталогов `bin` и `lib`;
- 2) отредактировать файл `~/.bashrc` в соответствии с примером рисунка 1.11;
- 3) запустить виртуальный терминал и выполнить проверку произведённых настроек с использованием утилиты `ij`;
- 4) перейти в каталог `~/databases` и изучить содержимое сценариев `startServerDB` и `stopServer`;
- 5) проверить запуск и остановку СУБД Apache Derby в сетевом режиме;
- 6) результаты работы отразить в личном отчёте.

1.2.3 Тестирование ПО сервера приложений Apache TomEE

Открытый для свободного использования сервер приложений Apache TomEE официально появился в 2012 году и представляет собой объединение двух проектов:

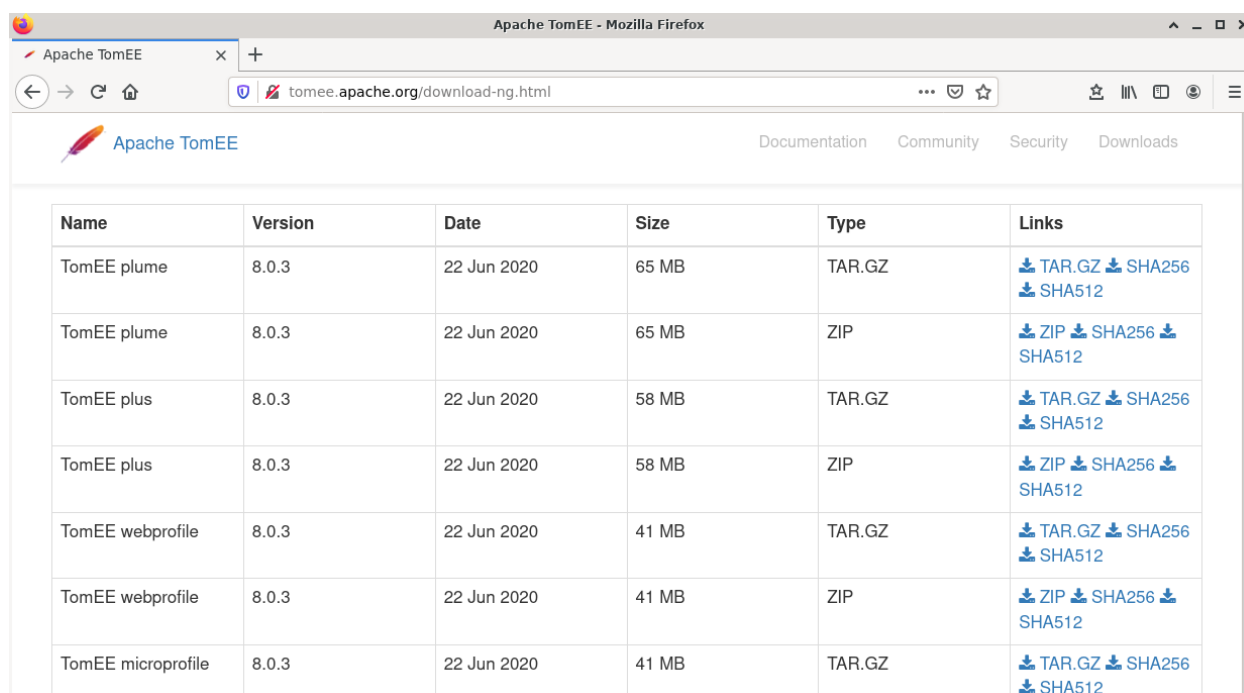
- 1) **Apache Tomcat** — высокопроизводительного web сервера и базового сервера поддерживающего контейнерные технологии (контейнер сервлетов);
- 2) **Java EE (Java Enterprise Edition)** — программной платформы языка Java для разработки приложений уровня предприятий.

Такая функциональная комбинация сервера позволяет создавать на его основе широкий круг распределённых систем.

Примечание — Выбор версии дистрибутива Apache TomEE должен соответствовать используемой базовой версии языка Java.

Хотя в ОС УПК АСУ уже установлен нужный дистрибутив сервера приложений, студент должен владеть технологией его выбора и установки:

- 1) сначала следует перейти на официальный сайт разработчика Apache TomEE по адресу <https://tomee.apache.org/> и открыть вкладку **Download**;
- 2) на странице **Download** следует выбрать ссылку на группу дистрибутивов, соответствующих используемой версии языка Java (в нашем случае — версия 8);
- 3) перейдя по нужной ссылке, мы увидим доступный для скачивания список дистрибутивов; в нашем это — варианты версий 8.0.3 показанные на рисунке 1.14.
- 4) выберем функционально наиболее полный вариант дистрибутива (**TomEE plume**) размером 65 МБ, доступный в виде сжатого файла **apache-tomee-8.0.3-plume.tar.gz** и скачиваем его любым способом.



Name	Version	Date	Size	Type	Links
TomEE plume	8.0.3	22 Jun 2020	65 MB	TAR.GZ	TAR.GZ SHA256 SHA512
TomEE plume	8.0.3	22 Jun 2020	65 MB	ZIP	ZIP SHA256 SHA512
TomEE plus	8.0.3	22 Jun 2020	58 MB	TAR.GZ	TAR.GZ SHA256 SHA512
TomEE plus	8.0.3	22 Jun 2020	58 MB	ZIP	ZIP SHA256 SHA512
TomEE webprofile	8.0.3	22 Jun 2020	41 MB	TAR.GZ	TAR.GZ SHA256 SHA512
TomEE webprofile	8.0.3	22 Jun 2020	41 MB	ZIP	ZIP SHA256 SHA512
TomEE microprofile	8.0.3	22 Jun 2020	41 MB	TAR.GZ	TAR.GZ SHA256 SHA512

Рисунок 1.14 — Страница с дистрибутивами Apache TomEE

Далее поступаем, как и ранее, с дистрибутивом Apache Derby: помещаем скачанный файл **apache-tomee-8.0.3-plume.tar.gz** в отдельный каталог и распаковываем его командой выражения (1.6).

```
sudo tar -xvfz apache-tomee1-8.0.3-plume.tar.gz
```

 (1.6)

Создаём новый файл дистрибутива Apache TomEE командой (1.7).

```
mksquashfs ./apache-tomee-8.0.3-plume apache-tomee-plume-803.sfs
```

 (1.7)

Помещаем файл **apache-tomee-plume-803.sfs** в каталог дистрибутива ОС УПК АСУ. При необходимости создаём каталоги **/opt/tomee** и **\$HOME/tomee**.

Примечание — Для пользователя *upk* файл *apache-tomee-plume-803.sfs* доступен через полный путь: */run/basefs/asu64upk/opt/apache-tomee-plume-803.sfs*.

Настройка среды ОС УПК АСУ для сервера приложений Apache TomEE.

В основе базовой технологии сервера Apache TomEE лежит контейнерная технология сервера Apache Tomcat. Фактически, сервер приложений TomEE и запускается как сервер Tomcat, активизируя прослушивание по умолчанию: порт **8080** и адрес *localhost*.

Примечание — Полная документация по всем аспектам использования сервера Apache TomEE находится по адресу: <http://tomee.apache.org/tomee-8.0/docs/>. В корне дистрибутива находится файл ***RUNNING.txt***, который описывает все аспекты конфигурации и запуска сервера. Им и воспользуемся при описании данного пункта.

При запуске сервера Apache TomEE используются следующие основные переменные среды ОС:

- 1) ***JAVA_HOME*** — каталог дистрибутива JDK;
- 2) ***JAVA_JRE*** — каталог дистрибутива JRE;
- 3) ***CATALINA_HOME*** — каталог дистрибутива TomEE, являющийся обязательным параметром при запуске сервера (в нашем случае — это каталог */opt/tomee*, куда должна монтироваться файловая система файла */run/basefs/asu64upk/opt/apache-tomee-plume-803.sfs*);
- 4) ***CATALINA_BASE*** — каталог для «персональных» настроек дистрибутива, не являющийся обязательным параметром при запуске сервера (в нашем случае — это каталог ***\$HOME/tomee***).

Для настройки среды ОС добавим в файл ***\$HOME/.bashrc*** переменные среды ОС для сервера, как это показано на рисунке 1.15.

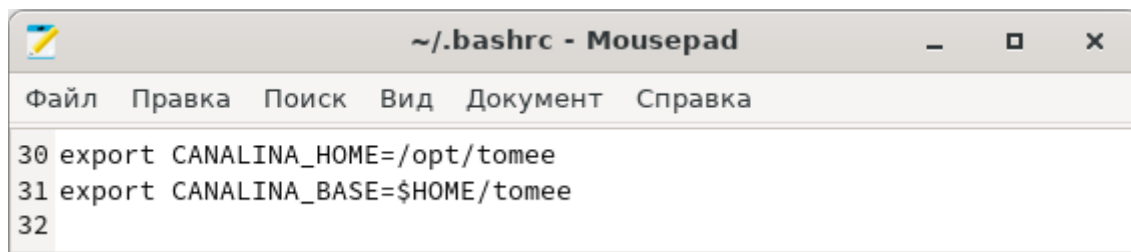


Рисунок 1.15 — Настройка переменных среды сервера Apache TomEE

Примечание — Добавленных переменных среды — вполне достаточно для тестирования запуска сервера приложений Apache TomEE.

Тестирование работоспособности ПО сервера приложений Apache TomEE.

После запуска сценария *~/bin/mountTomee* дистрибутив сервера монтируется в каталог */opt/tomee*. С другой стороны каталог ***\$HOME/tomee*** содержит те же файлы и каталоги, что и каталог дистрибутива, кроме каталога *lib*, который должен быть пустым. В этом можно убедиться, сравнив содержимое рисунка 1.16 и каталога */opt/tomee*.

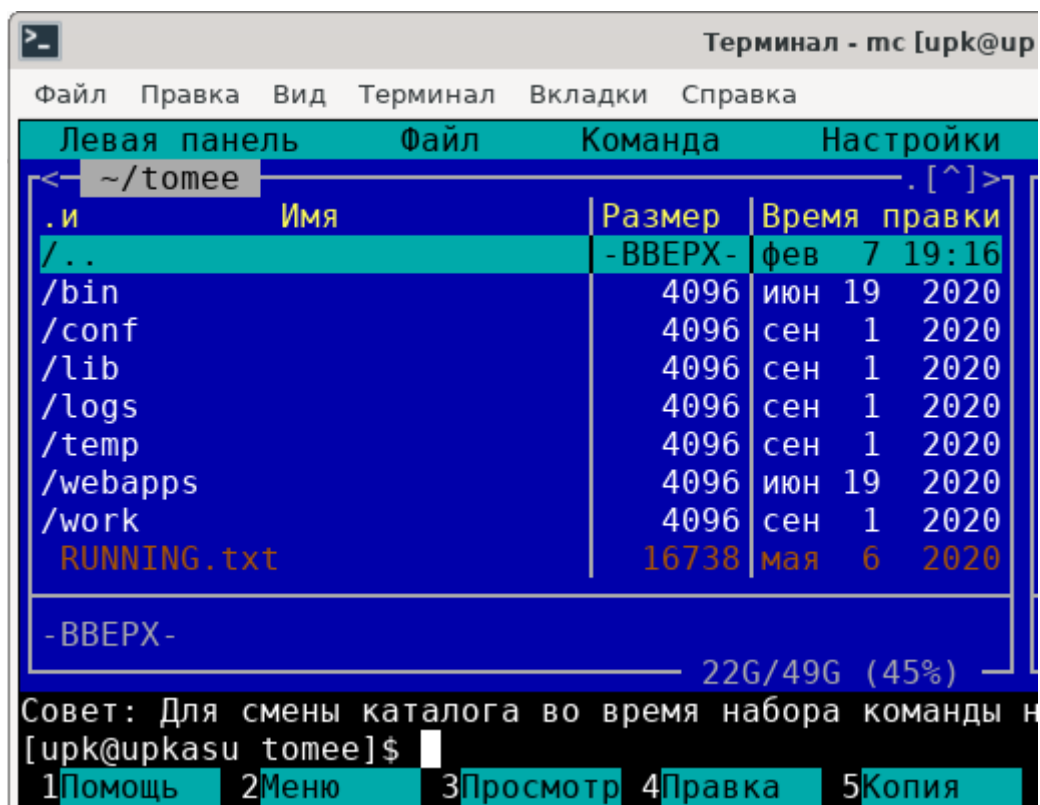


Рисунок 1.16 — Содержимое каталога «персональных» настроек дистрибутива сервера Apache TomEE

Перейдя в каталог `~/tomee/bin`, мы увидим множество сценариев (см. рисунок 1.7), среди которых нам потребуются сценарии `startup.sh` и `shutdown.sh`.

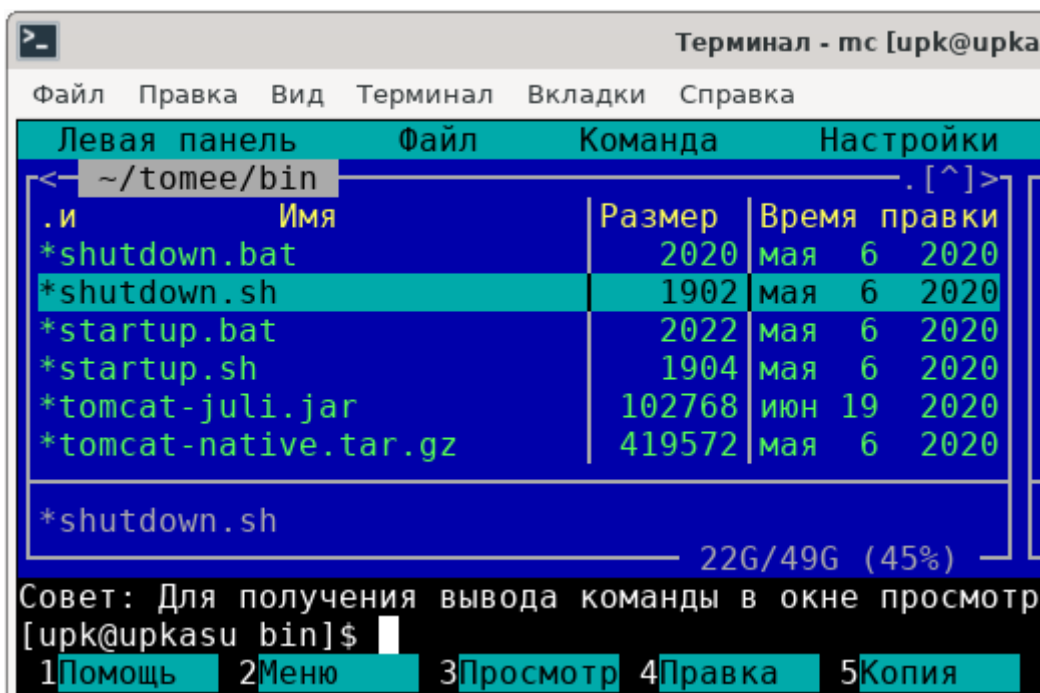
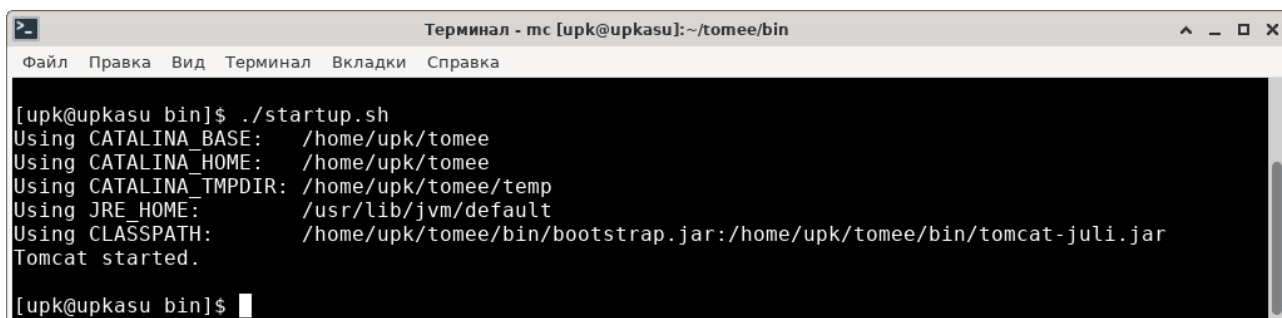


Рисунок 1.17 — Содержимое каталога \$HOME/tomee/bin

Если из каталога ***\$HOME/tomee/bin*** выполнить команду (1.8), то сервер приложений запустится, как показано на рисунке 1.18.

./bin/startup.sh (1.8)



```
Терминал - mc [upk@upkasu]:~/tomee/bin
Файл  Правка  Вид  Терминал  Вкладки  Справка

[upk@upkasu bin]$ ./startup.sh
Using CATALINA_BASE:   /home/upk/tomee
Using CATALINA_HOME:   /home/upk/tomee
Using CATALINA_TMPDIR: /home/upk/tomee/temp
Using JRE_HOME:        /usr/lib/jvm/default
Using CLASSPATH:       /home/upk/tomee/bin/bootstrap.jar:/home/upk/tomee/bin/tomcat-juli.jar
Tomcat started.

[upk@upkasu bin]$
```

Рисунок 1.18 — Пример запуска сервера Apache TomEE

При старте сервер приложений показывает основные системные переменные среды, с которыми он запущен.

Если теперь браузером подключиться к адресу <http://localhost:8080/>, то появится базовая страница сервера Apache TomEE, показанная на рисунке 1.19.

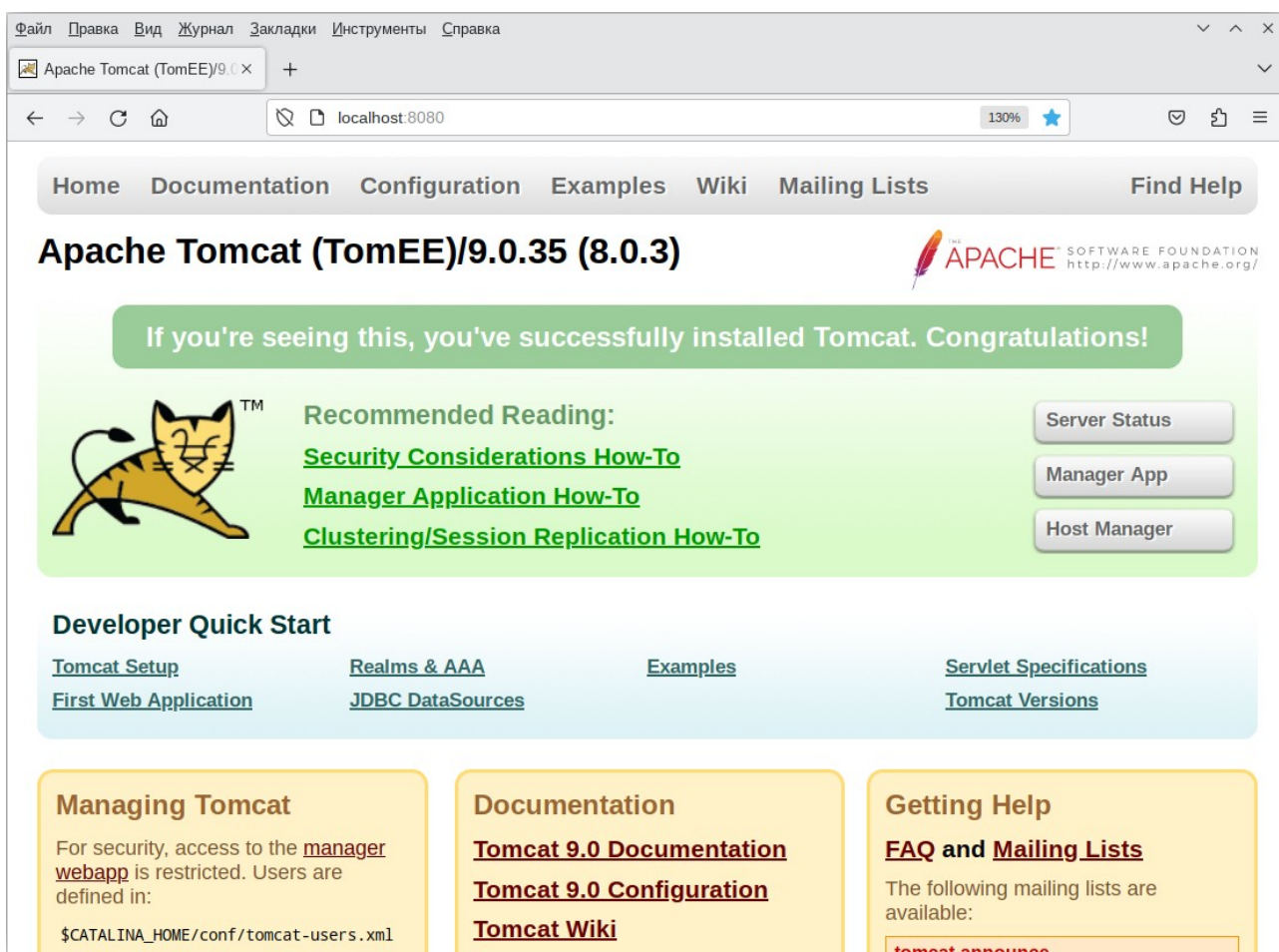


Рисунок 1.19 — Соединение с запущенным сервером Apache TomEE

Данный рисунок показывает, что сервер приложений Apache TomEE 8.0.3 основан на сервере Apache Tomcat версии 9.0.35. Зелёным цветом выделены ссылки на общие положения по безопасности и дополнительной конфигурации сервера. Завершение работы сервера можно выполнить командой выражения (1.9).

`./bin/shutdown.sh` (1.9)

Примечание - Дополнительные сведения и правила использования сервера Apache TomEE указываются в конкретных лабораторных работах.

Четвёртое учебное задание:

- 1) подключить к системе дистрибутив ПО Apache TomEE, перейти в каталог `/opt/tomee` и исследовать содержимое каталогов *bin* и *lib*;
- 2) отредактировать файл `~/.bashrc` в соответствии с примером рисунка 1.15;
- 3) перейти в каталог `~/tomee/bin` и запустить сервер Apache TomEE;
- 4) запустить браузер подключиться к серверу по адресу `http://localhost:8080/`, чтобы убедиться в работоспособности ПО Apache TomEE;
- 5) отразить результаты работы в личном отчёте;
- 6) закрыть окно браузера и остановить сервер Apache TomEE.

1.2.4 Тестирование ПО среды разработки Eclipse EE

Появившись в 2001 году, интегрированная среда разработки (IDE) Eclipse была предназначена для разработки программного обеспечения на языке Java для корпорации IBM. С тех пор эта среда претерпела множество изменений и сейчас все разработки (проекты) этой IDE координируются некоммерческой организацией Eclipse Foundation.

Примечание — В частности в среде ОС УПК АСУ установлен проект Eclipse CDT, предназначенный для программирования на платформах языков C/C++, что необходимо для проведения занятий по дисциплине «Операционные системы».

Непосредственно для изучаемой дисциплины необходим проект IDE Eclipse EE (Eclipse Enterprise Edition), поддерживающий развитие платформы языка Java EE (J2EE). Этот проект обеспечивает разработку систем РВС, но его необходимо устанавливать опционно, как и проекты Apache Derby и Apache TomEE, чтобы он не конфликтовал с уже установленным дистрибутивом Eclipse CDT. Рассмотрим как это было сделано для дистрибутива ОС УПК АСУ.

Если зайти на официальный сайт Eclipse Foundation, то можно потеряться среди различных предложений по установке проектов. Это затрудняет нужный выбор. Поэтому лучше сразу перейти по нужному нам адресу <https://www.genuitec.com/eclipse-packages/> и выбрать дистрибутив Eclipse EE, как это показано на рисунке 1.20.

Примечание — Обратите внимание, что на рисунке 1.20 показан дистрибутив версии 2021-06.

На середину 2020 года имелся дистрибутив версии 2020-06, который после нажатия ссылки «**Download**» был скачан как файл `eclipse-jee-2020-06-R-linux-gtk-x86_64.tar.gz`.

После преобразований, аналогичных выражениям (1.6) и (1.7), был получен дистрибутив Eclipse EE, доступный пользователю *upk* как файл */run/basefs/asu64upk/opt/eclipse-jee-2020.sfs*.

Этот дистрибутив и используется при выполнении лабораторных работ.

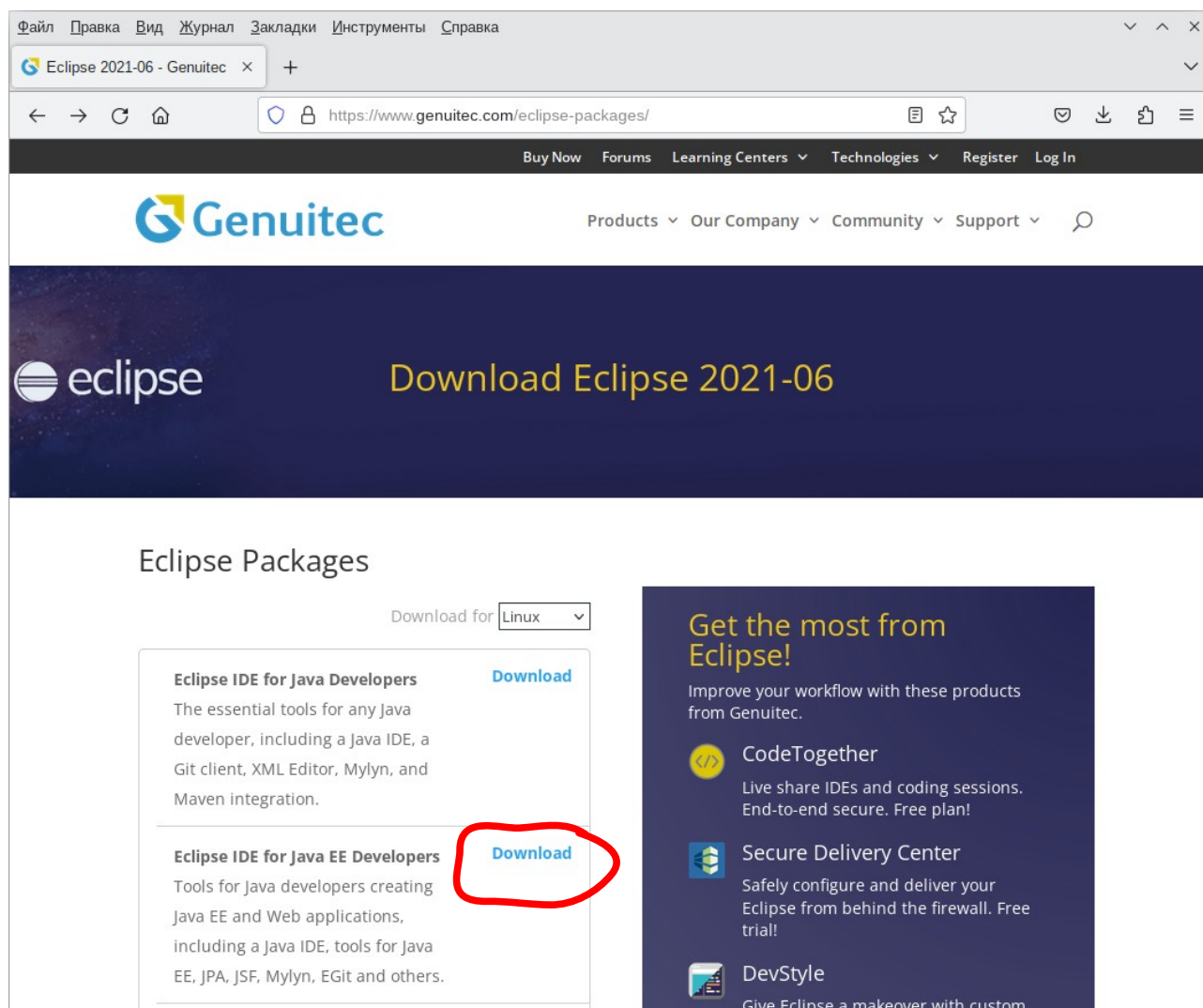


Рисунок 1.20 — Выбор дистрибутива Eclipse EE

Тестирование работоспособности ПО среды разработки Eclipse EE.

Тестирование работоспособности ПО Eclipse EE осуществляется в два этапа:

- 1) подключение дистрибутива среды разработки;
- 2) запуск самой среды разработки.

Первый этап осуществляется посредством запуска сценария *~/bin/mountEclipseEE*, в результате чего дистрибутив среды разработки монтируется в каталог */opt/eclipseEE*.

Перейдя в каталог */opt/eclipseEE*, убеждаемся, что подключение дистрибутива прошло успешно, как это показано на рисунке 1.21.

Второй этап тестирования осуществляется посредством запуска специального значка с именем «EclipseEE», но прежде чем это сделать нужно убедиться в правильной его настройке, как это показано на рисунке 1.22.

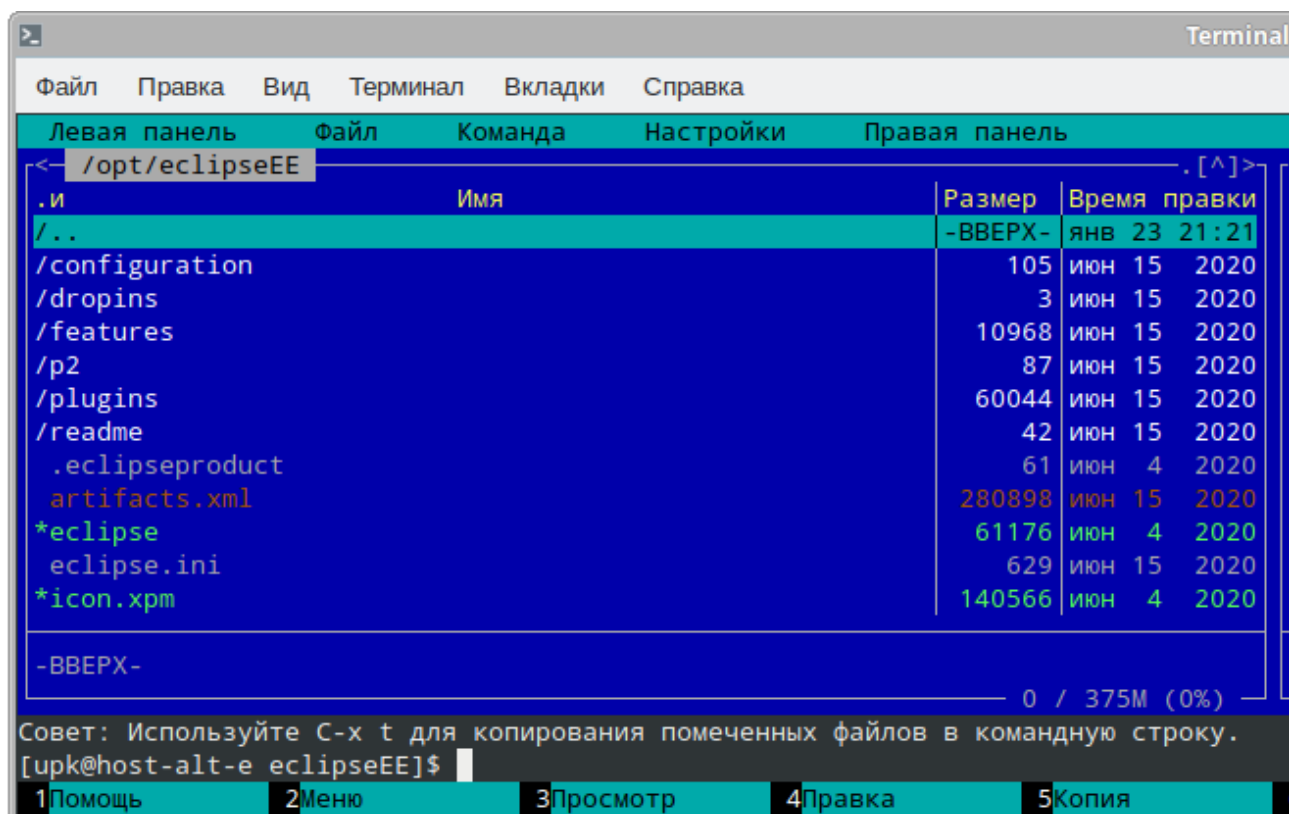


Рисунок 1.21 — Проверка подключения дистрибутива Eclipse EE

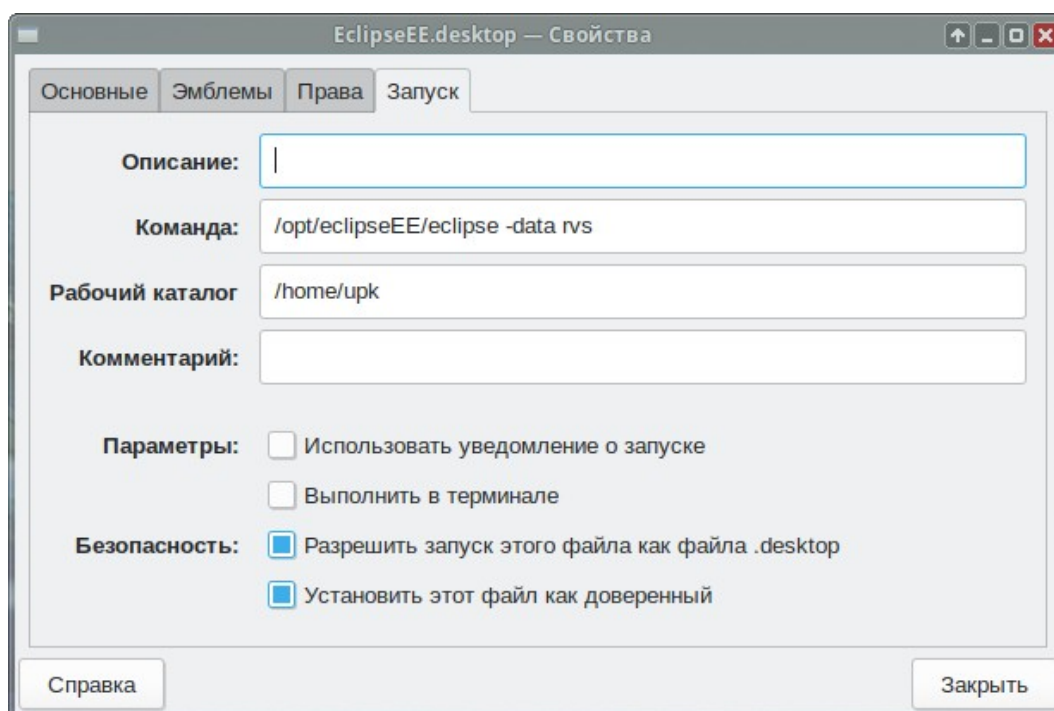


Рисунок 1.22 — Проверка настроек значка запуска «EclipseEE»

Примечание — Чтобы вызвать окно свойств значка запуска, следует: установить на него курсор мышки и активировать её правую кнопку, а затем перейти на вкладку «Запуск».

При первом запуске среды разработки Eclipse EE система IDE загрузится с заставкой «Welcome», как это показано на рисунке 1.23. Активация кнопки «Workbench», размещённой в правом верхнем углу, удаляет это окно приглашения.

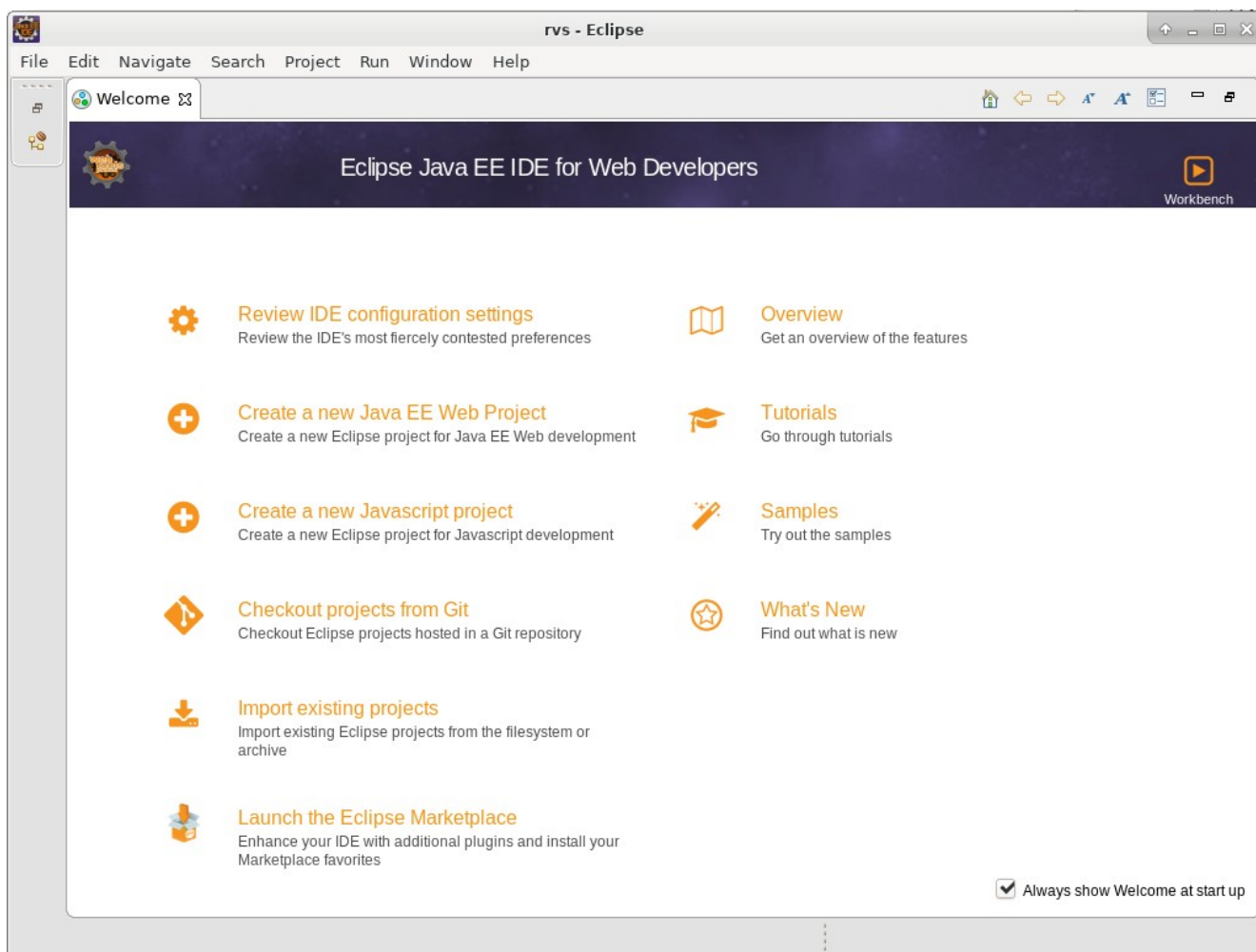


Рисунок 1.23 — Первый запуск среды разработки Eclipse EE

Правильно запущенная среда разработки **Eclipse EE** должна создать в рабочей директории **\$HOME** пользователя **upk** каталог **~/rvs** с необходимыми первоначальными настройками. Студент должен непосредственно убедиться в правильности создания этого рабочего каталога.

Пятое учебное задание:

- 1) подключить к системе дистрибутив ПО Apache TomEE и Eclipse EE;
- 2) провести запуск среды разработки Eclipse EE и убедиться, что она запускается как это показано на рисунке 1.23;
- 3) отразить результаты выполнения задания в личном отчёте;
- 4) завершить выполнение лабораторной работы, закрыв все окна, остановив все сервера и выполнив архивирование личной рабочей области пользователя **upk**.

2 ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ЯЗЫКА JAVA

Благодаря высокой переносимости между операционными системами (ОС) и аппаратными средствами вычислительной техники (ЭВМ) язык Java получил большую популярность среди разработчиков распределённых вычислительных систем (РВС). Этому процессу также способствовали быстрое развитие инструментальных средств языка, ориентированных на практическую реализацию различных задач конкретных приложений.

Общая учебная цель второго раздела лабораторных занятий — практическое освоение инструментальных средств языка Java ориентированных на решение набора стандартных прикладных задач, решаемых каждым специалистом в области информационных технологий.

В пределах данной темы должны быть выполнены пять лабораторных работ, которые последовательно ориентированы на получение соответствующих практических навыков программирования.

Работа №3 «Базовые инструментальные средства языка Java» — продолжение тематики первых двух работ, связанных с манипулированием программными объектами изучаемого языка: исходными текстами программ, их компиляцией, размещением, созданием архивных библиотечных форм, запуском на исполнение, а также созданием среды операционных систем, для последующей эксплуатации реализованных приложений.

Работа №4 «Классы, интерфейсы и методы языка Java» — практическое освоение базовых синтаксических конструкций языка Java, используемых программными средствами языка для реализации приложений.

Работа №5 «Ввод/вывод языка Java» — практическое освоение широкого спектра средств ввода/вывода языка, обеспечивающих реальную манипуляцию с объектными типами самого языка и сохранение этих объектов в файловых системах ОС.

Работа №6 «Сокеты и сетевое ПО языка Java» — практическое освоение сетевых базовых средств языка Java, составляющих основу реализации распределённых вычислительных систем (РВС).

Работа №7 «Технология работы с СУБД Derby» — практическое освоение навыков работы с СУБД Derby, которое необходимо студенту для реализации простейших информационных систем. Этой работой завершается базовое обучение студента языку Java, что обеспечивает ему возможность последующего изучения технологий распределённых систем (РВС). В частности, учебный пример приложения, реализуемый студентом в данной работе, используется в лабораторных работах последующих тем изучаемой дисциплины.

2.1 Лабораторная работа №3.

Базовые инструментальные средства языка Java

Данная лабораторная работа предназначена для практического освоения инструментальных средств языка Java, обеспечивающих его функционирование в среде операционной системы ОС УПК АСУ.

Учебная цель данной работы — изучение технологии манипулирования программными *мета-объектами* языка Java, обеспечивающими:

- 1) компиляцию исходных текстов программ;
- 2) создание запускаемых в среде ОС приложений;
- 3) запуск готовых приложений на исполнение.

Практическая цель работы — практическое закрепление проектной работы с утилитами *java*, *javac* и *jar*, а также интегрированной средой разработки *Eclipse EE*, использование которых студент уже должен был изучить по тексту подраздела 2.1 учебного пособия [1].

Общее учебное задание — достичь учебной и практической целей лабораторной работы посредством реализации учебного примера приложения *Example1* в форме проекта с именем *proj1*.

Рекомендуемые операции и действия для выполнения данной работы изложены в следующей последовательности пунктов:

- 1) организация командной среды разработки проекта *proj1*;
- 2) компиляция и тестирование приложения *Example1*;
- 3) создание и запуск проекта *proj1.jar*;
- 4) реализация проекта *proj1* в инструментальной среде IDE Eclipse EE.

2.1.1 Организация командной среды разработки проекта *proj1*

Результаты выполнения предыдущей лабораторной работы и общий учебный материал подраздела 2.1 учебного пособия [1] показывают, что инструментальная среда разработки приложений на языке Java требуют задания множества параметров среды ОС, которые должны правильно указывать местоположение различных источников информации: дистрибутива самого языка Java, местоположение исходных текстов программ и результатов их обработки.

Примечание — Использование для проектной деятельности файла *~/.bashrc* является нежелательным по причине возможного разрушения программной среды пользователя.

Правильной организацией командной среды проекта *proj1* будет создание отдельного сценария *proj1.sh*, в котором следует разместить все нужные для реализации проекта параметры среды ОС.

Примечание — Общее правило размещения ПО в средах ОС UNIX и Linux предполагает, что: запускаемые программы размещаются в каталогах *bin*; исходные тексты программ — в каталогах *src*; различные библиотеки — в каталогах *lib*.

Следуя общим рекомендациям размещения ПО ОС:

- 1) создадим (если они не созданы) в рабочей среде пользователя **upk** каталоги **~/bin**, **~/lib** и **~/src/rvs**;
- 2) в каталог **~/bin** следует поместить сценарий **proj1.sh**;
- 3) в каталог **~/src/rvs** следует поместить файл **Example1.java**.

Результат произведённого размещения файлов показан на рисунке 2.1.

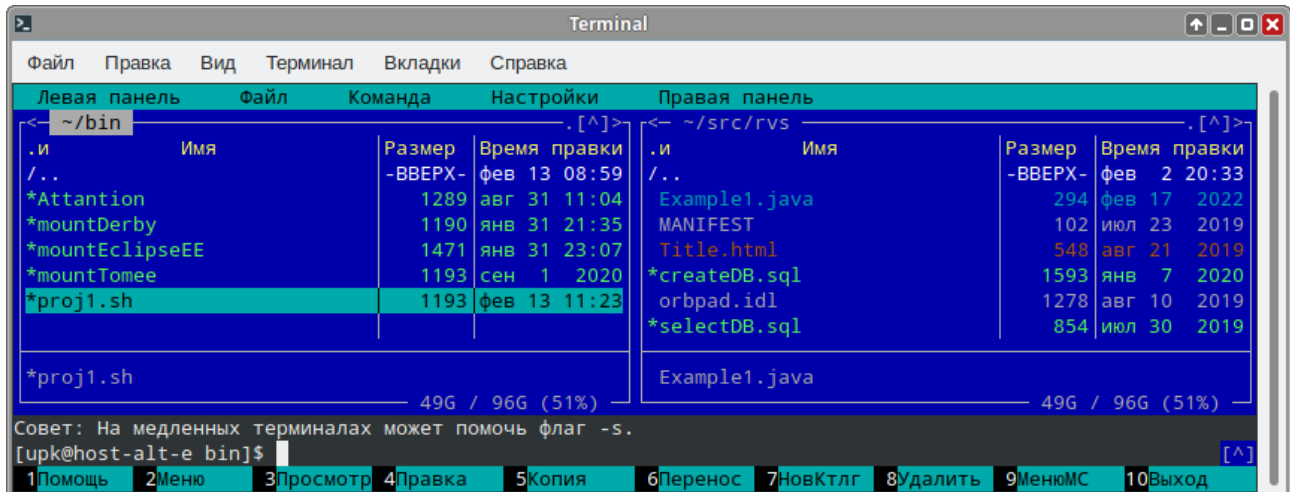


Рисунок 2.1 — Начальный вариант размещения файлов проекта proj1

Преимущество использования отдельного файла проекта **~/bin/proj1.sh** состоит в возможности запуска его без опасности повредить файл **~/bashrc** и отлаживать команды тестирования проект. Начальное содержимое этого файла показано на рисунке 2.2.

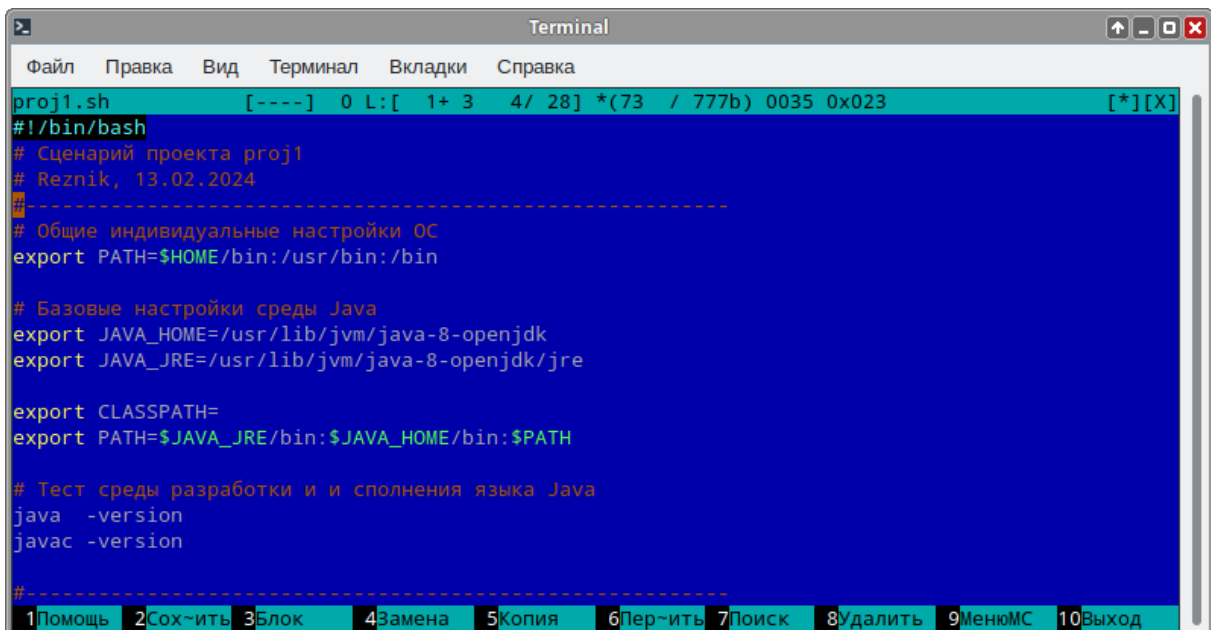


Рисунок 2.2 — Начальное содержимое файла `~/bin/proj1.sh`

Следует запустить файл **~/bin/proj1.sh** на исполнение и проверить правильную настройку среды запуска утилит **java** и **javac**.

Примечание — Запуск сценария `~/bin/poroj1.sh` в среде ОС УПК АСУ покажет правильную настройку утилит `java` и `javac`, что уже было показано в предыдущей работе на рисунке 1.8. Для случая использования других ОС этот вопрос следует исследовать отдельно.

Для примера рассмотрим дистрибутив ОС «Альт Образование 10.2», в среде которого пишется данные методические указания, то перейдя в каталог `/usr/lib/jvm`, где установлены дистрибутивы инструментальных средств языка Java, мы увидим картину показанную на рисунке 2.3.

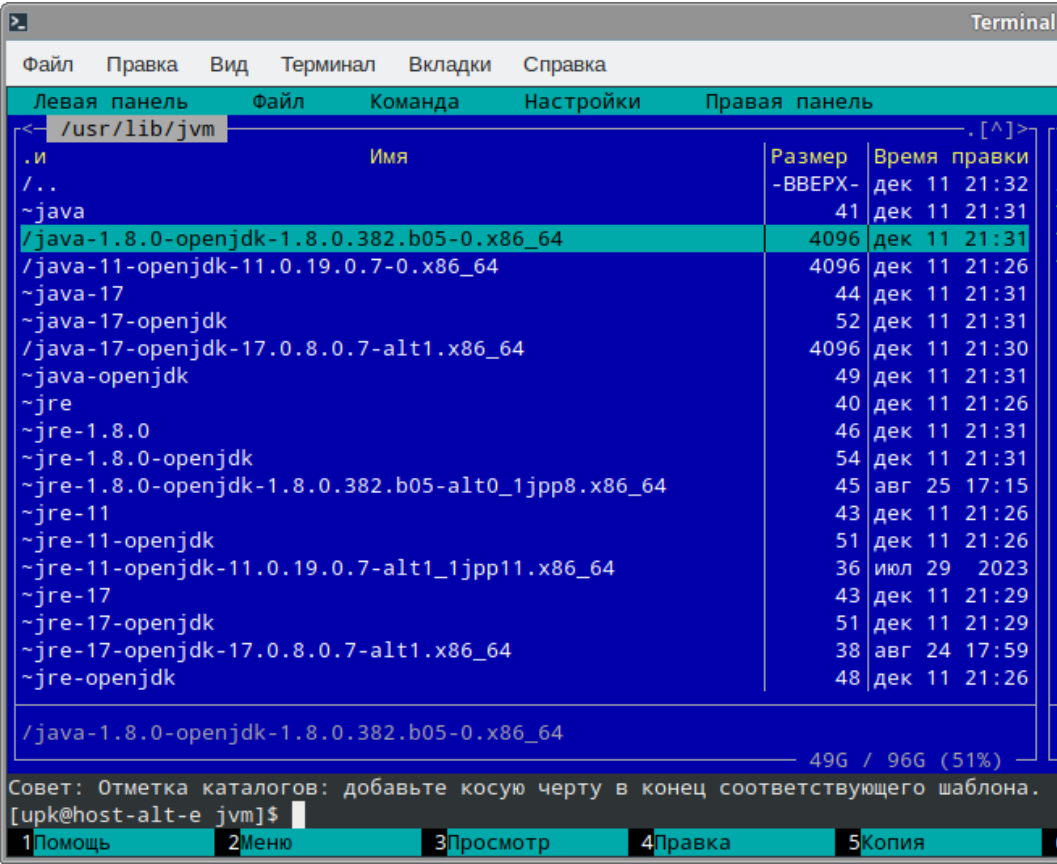


Рисунок 2.3 — Дистрибутивы Java ОС «Альт Образование 10.2»

Нужная нам версия `java-1.8.0-openjdk...`, выделенная на рисунке 2.3 курсором, содержит только ПО JRE, что хорошо видно на рисунке 2.4.

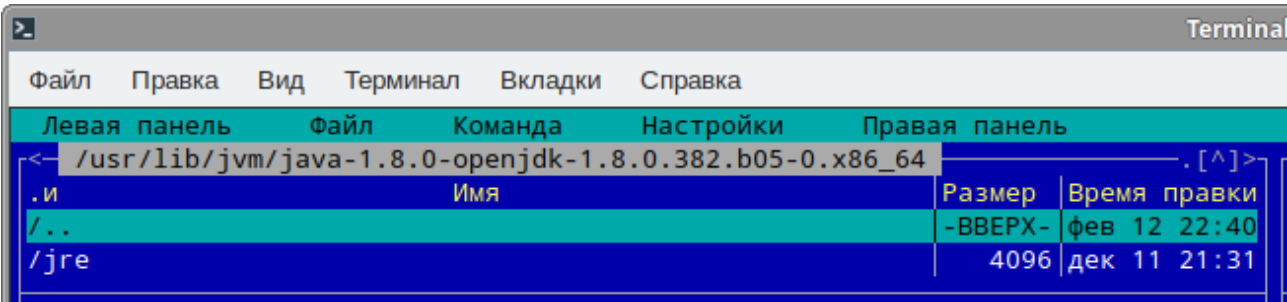


Рисунок 2.4 — Содержимое каталога `java-1.8.0-openjdk...`

Данная ситуация соответствует случаю, когда в среде ОС Linux установлены три версии дистрибутивов языка Java: 8, 11 и 17. Причём:

- 1) дистрибутив JRE имеется для всех версий;
- 2) полный дистрибутив JDK имеется только для версии 17.

В сложившейся ситуации содержимое файла `~/bin/poroj1.sh` следует представить в виде, который показан на рисунке 2.5.

Рисунок 2.5 — Новое содержимое файла `~/bin/proj1.sh`

Теперь запуск сценария покажет результат, отражённый на рисунке 2.6.

Рисунок 2.6 — Результат запуска сценария `~/bin/proj1.sh` на исполнение

Таким образом, мы имеем ситуацию, когда среда исполнения и среда разработки языка Java имеют разные версии.

Примечание — Ситуации с разными версиями среды разработки и среды исполнения языка Java являются типичными при создании и эксплуатации ПО приложений.

2.1.2 Компиляция и тестирование приложения Example1

Тестовое приложение *Example1* представлено файлом `~/src/rvs/Example1.java`, содержимое которого показано на рисунке 2.7.

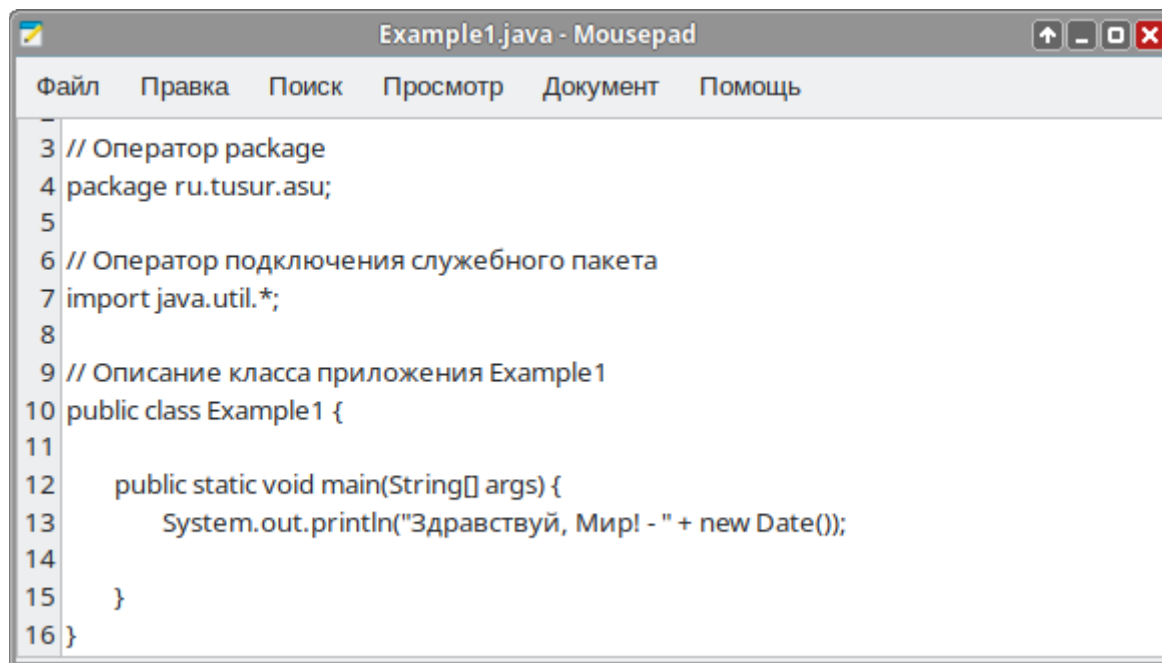


Рисунок 2.7 — Содержимое файла тестового приложения Example1

Рассматриваемый файл исходного текста приложения *Example1* содержит все необходимые элементы запускаемого приложения языка Java: операторы *package*, *import* и *class*, а также метод *main(...)*. Если перейти в каталог `~/src/rvs` и выполнить команду (2.1), появится файл *Example1.class*, являющийся результатом компиляции файла *Example.java*, что наглядно показано на рисунке 2.8.

`javac Example1.java` (2.1)

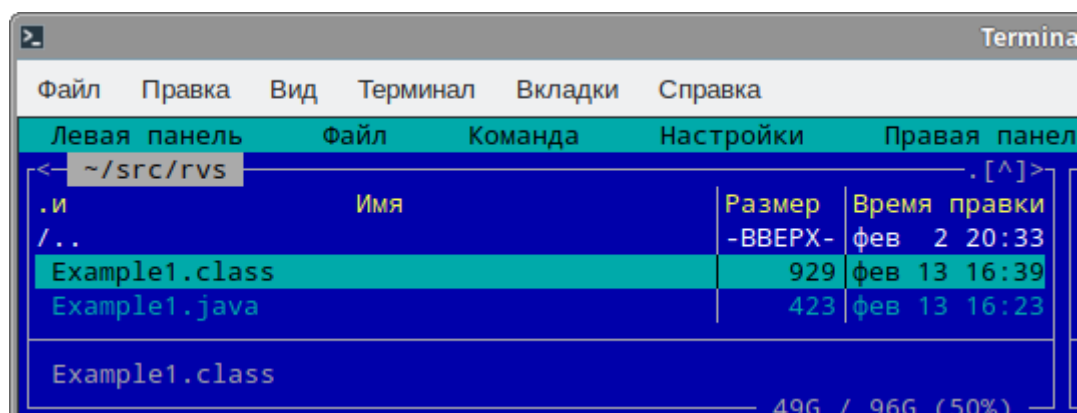
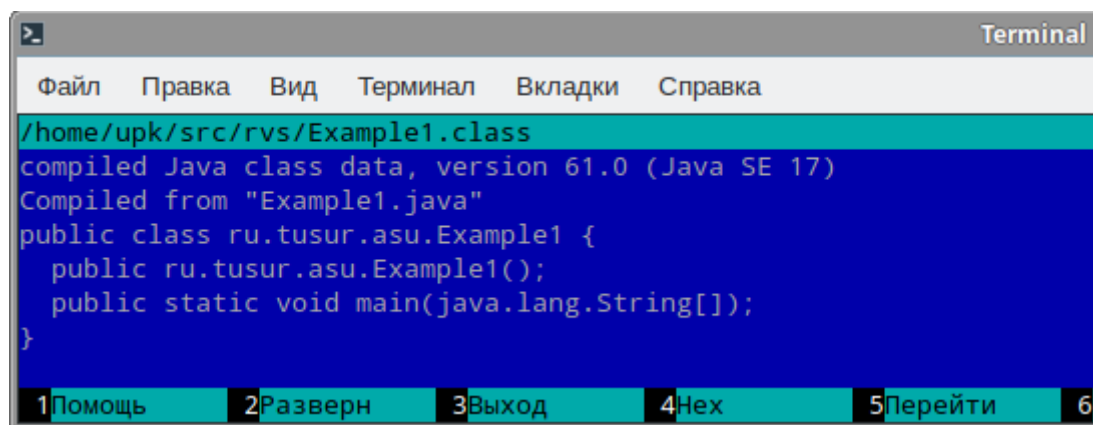


Рисунок 2.8 — Результат выполнения команды (2.1) в каталоге `~/src/rvs`

Выделим *Example1.class* и нажмём клавишу *F3*. Получим результат рисунка 2.9.



```
/home/upk/src/rvs/Example1.class
compiled Java class data, version 61.0 (Java SE 17)
Compiled from "Example1.java"
public class ru.tusur.asu.Example1 {
    public ru.tusur.asu.Example1();
    public static void main(java.lang.String[]);
}
```

Рисунок 2.9 — Просмотр содержимого файла Example1.class

Уже первая строка рисунка 2.9 показывает, что компиляция исходного текста приложения проведена для дистрибутива Java версии 17, поэтому она не будет запускаться JRE Java версии 8.

Чтобы получить нужный вариант результата компиляции исходных текстов языка Java, следует запустить команду выражения (2.2), разодраться в опциях утилиты *javac* и правильно настроить её командную строку запуска.

javac -help (2.2)

Первое учебное задание:

- 1) выполнить работы по пункту 2.1.1 данного подраздела методических указаний;
- 2) используя команду выражения (2.2) разобраться с опциями утилиты *javac*;
- 3) дописать сценарий *~/bin/proj1.sh*, используя вызов утилиты *javac*, которая берёт файл *Example1.java* из каталога *~/src/rvs*, компилирует его и результат компиляции помещает в каталог *~/lib/proj1*;
- 4) запустить сценарий *~/bin/proj1.sh* на исполнение и убедиться, что он откомпилирован под нужную версию дистрибутива утилиты *java*;
- 5) отразить результаты работы в личном отчёте.

Примечание — Далее по тексту приведено описание выполнения первого задания для дистрибутива ОС «Альт Образование 10.2».

Учитывая, что работы по пункту 2.2.1 уже — выполнены, запустим в терминале команду выражения 2.2 и выполним следующие действия:

- 1) по результатам изучения множества опций утилиты *javac* выпишем наиболее интересные опции, которые отразим в таблице 2.1;
- 2) по результатам анализа таблицы 2.1 добавим в файл *~/bin/proj1.sh* строку компиляции файла *Example1.java*, как это показано на рисунке 2.10;
- 3) запустим файл *~/bin/proj1.sh* на исполнение и убедимся в нормальном его завершении;
- 4) перейдём в каталог *~/lib/proj1/ru/tusur/asu* и убедимся, что там появился файл откомпилированного класса *Example1.class*.

Таблица 2.1 — Наиболее важные опции запуска утилиты `javac` версии 17

Опция	Семантика опции
<code>--class-path <path></code> <code>-classpath <path></code> <code>-cp <path></code>	Три варианта задания списка подключаемых файлов классов пользователей. Параметр <code><path></code> - список объектов, разделённых символом двоеточия. Этот путь к классу переопределяет путь к пользовательскому классу в переменной среды <code>CLASSPATH</code> .
<code>-d directory</code>	Устанавливает каталог назначения (или выходной каталог класса) для файлов классов. Если класс является частью пакета, то <code>javac</code> помещает файл класса в подкаталог, который отражает имя модуля (если применимо) и имя пакета. Каталог и все необходимые подкаталоги будут созданы, если они еще не существуют.
<code>--release release</code>	Компилирует исходный код в соответствии с правилами языка программирования Java для указанной версии Java SE, генерируя файлы классов, предназначенные для этой версии. Исходный код компилируется с использованием объединенного API Java SE и JDK для указанной версии. Доступные значения release : 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17.
<code>-s directory</code>	Указывает каталог, используемый для размещения сгенерированных исходных файлов. Если класс является частью пакета, то компилятор помещает исходный файл в подкаталог, который отражает имя модуля (если необходимо) и имя пакета. Каталог и все необходимые подкаталоги будут созданы, если они еще не существуют.
<code>--source-path path</code> <code>-sourcepath path</code>	Указывает, где найти исходные файлы. За исключением случаев компиляции нескольких модулей вместе, это путь к исходному коду, используемый для поиска определенных классов или интерфейсов.
<code>-verbose</code>	Выводит сообщения о том, что делает компилятор. Сообщения включают информацию о каждом загруженном классе и каждом скомпилированном исходном файле.

```

proj1.sh  [-----]  0 L:[ 12+ 3  15/ 33] *(499 /1032b) 0035 0x0[*][X]
export CLASSPATH=
export PATH=$JAVA_JRE/bin:$JAVA_HOME/bin:$PATH

# Тест среды разработки и и сполнения языка Java
java -version
javac -version

# Запуск утилиты javac
javac --release 8 -d $HOME/lib/proj1 -verbose $HOME/src/rvs/Example1.java

#-----
1По-щ 2Со-ть 3Блок 4Замена 5Копия 6Пе-ть 7Поиск 8Уда-ть 9МенюМС10Выход
  
```

Рисунок 2.10 — Добавление команды компиляции `javac` в файл `~/bin/proj1.sh`

Откроем на просмотр вновь созданный файл **Example1.class**, как это показано на рисунке 2.11, и убедимся, что он откомпилирован под JRE версии 8.

В завершение добавим в файл `~/bin/proj1.sh` строку выражения (2.3), выполняющего команду запуска приложения **Example1**.

`java -cp $HOME/lib/proj1 ru.tusur.asu.Example1` (2.3)

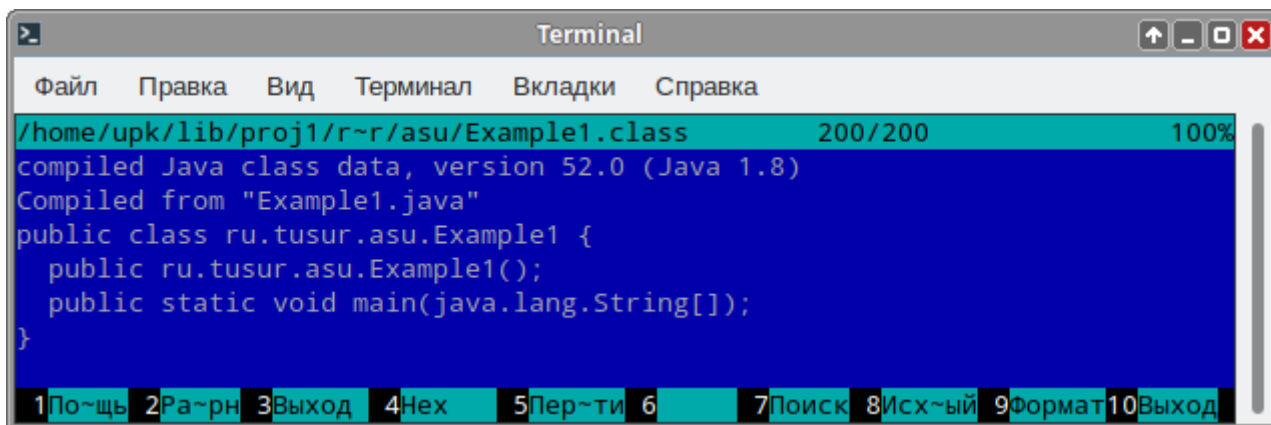


Рисунок 2.11 — Просмотр содержимого файла Example1.class в каталоге ~/lib/proj1/ru/tusur/asu

Конечный результат запуска сценария ~/bin/proj1.sh демонстрируется рисунком 2.12.

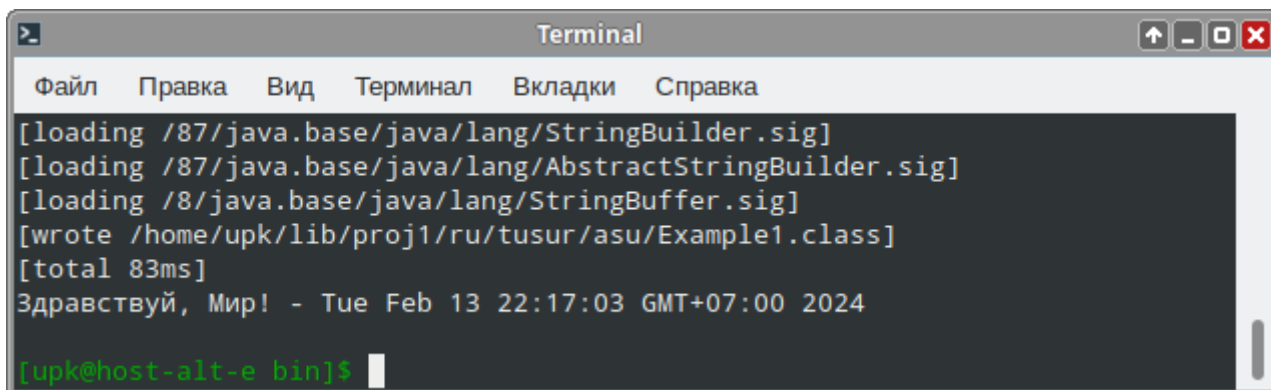


Рисунок 2.12 — Конечный результат запуска сценария ~/bin/proj1.sh

Результат показанный на рисунке 2.12 подтверждает, что первое учебное задание — выполнено!

2.1.3 Создание и запуск проекта proj1.jar

Конечным результатом любой разработки проекта является создание удобной для распространения формы приложения. Базовые инструментальные средства языка Java предоставляют для этих целей утилиту *jar*, которая обеспечивает создание и модификацию проектов приложений, обычно представляющих собой набор файлов некоторого дерева каталогов, в виде архивных файлов формата *zip*.

С прикладной точки зрения архивы языка Java:

- 1) представляют собой файлы *zip* с расширением *.jar*;
- 2) размещают классы языка Java в дереве каталогов согласно оператору *package*;
- 3) подразделяются на библиотечные архивы и запускаемые приложения.

Учебная цель данного пункта — практическое изучение утилиты *jar* в плане создания архивного файла *proj1.jar* проекта *proj1*, обеспечивающего запуск учебного приложения *Example1*.

Реализацию учебной цели проведём в два этапа:

- 1) создадим запускаемый на исполнение архив проекта *proj1* в каталоге *~/lib*;
- 2) проведём тестирование запуска архивного файла *proj1.jar*.

Создание архивного файла *proj1.jar*.

Выполним в командной строке терминала команду выражения (2.4) и выпишем в таблицу 2.2 необходимые для создания архива опции утилиты *jar*.

jar -help (2.4)

Таблица 2.2 — Наиболее важные опции утилиты *jar* для создания архивов

Опция	Семантика опции
--create -c	Создание архива.
-C DIR	Устанавливает каталог соответствующий вершине дерева каталогов архивируемого приложения.
-f FILE --file=FILE	Задаёт имя файла архива.
-v	Генерирует протокол работы утилиты <i>jar</i> на стандартный канал вывода сообщений.
--manifest=FILE -m FILE	Указывает файл манифеста архива.
-e CLASSNAME --main-class=CLASSNAME	Указывает имя класса архива, запускающего приложение.

Теперь, используя общий формат команды *jar* определённый выражением (2.5), запишем в конец файла *~/bin/proj1.sh* команду создания архива *~/lib/proj1.jar*, как это показано на рисунке 2.13.

jar <ОПЦИИ> -C <КАТАЛОГ> <ФАЙЛЫ> (2.5)

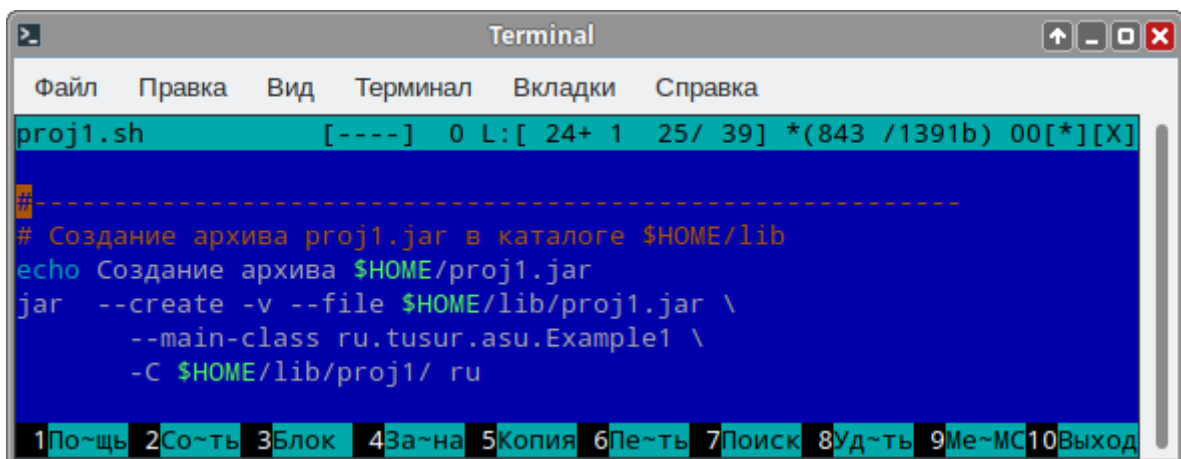


Рисунок 2.13 — Запись в файле *~/bin/proj1.sh* на создание архива *~/lib/proj1.jar*

Запустив сценарий *~/bin/proj1.sh* на исполнение, мы видим:

- 1) появление архива *proj1.jar* в каталоге *~/lib* (см. рисунок 2.14);
- 2) структуру каталогов корня архива *proj1.jar* (см. рисунок 2.15).

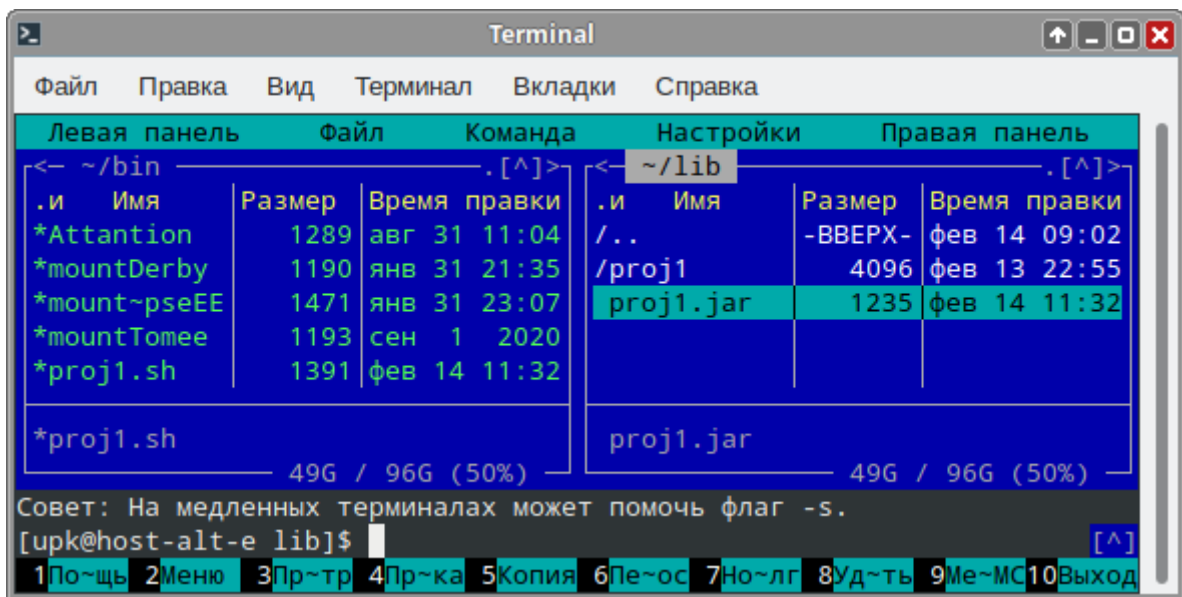


Рисунок 2.14 — Наличие файла архива в каталоге `~/lib`

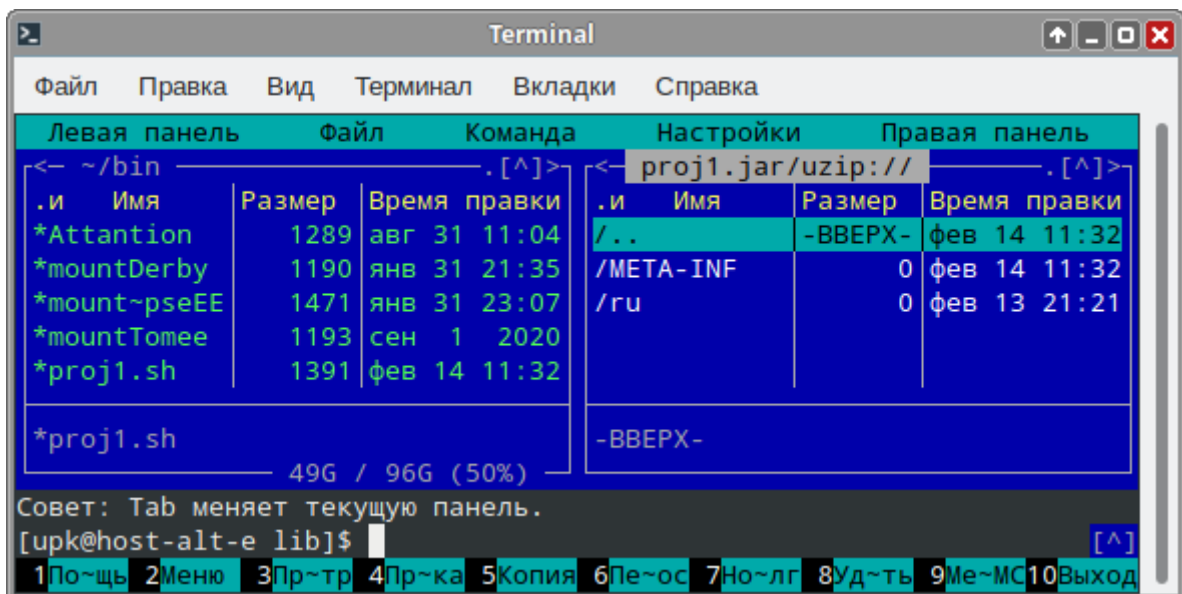


Рисунок 2.15 — Структура верхнего уровня архива `proj1.jar`

Тестирование запуска архивного файла `proj1.jar`.

Созданный нами архив `proj1.jar` предназначен для запуска на исполнение приложения *Example1*, реализованного в виде класса `ru.tusur.asu.Example1.class`.

Утилита `java` обеспечивает нам два основных варианта запуска приложения *Example1*:

- 1) прямой вызов архива с использованием опции `-jar`;
- 2) косвенный вызов класса `Example1.class` из созданного архива.

Первый вариант запуска приложения *Example1* осуществляется командой заданной выражением (2.6).

`java -jar $HOME/lib/proj1.jar` (2.6)

Второй вариант запуска приложения *Example1* предполагает предварительное задание значения переменной среды **CLASSPATH**, например, как это показано выражением (2.7), и последующее использование утилиты *java* согласно выражению (2.8).

export CLASSPATH=\$HOME/lib/proj1.jar (2.7)

java ru.tusur.asu.Example1 (2.8)

Второе учебное задание:

- 1) выполнить работы по пунктам 2.1.2 и 2.1.3 данного подраздела методических указаний;
- 2) дописать сценарий *~/bin/proj1.sh*, используя вызов утилиты *jar* для создания архива *~/lib/proj1.jar*;
- 3) дописать сценарий *~/bin/proj1.sh*, используя различные вызовы утилиты *java* для запуска приложения *Example1*;
- 4) провести тестирование запуска приложения *Example1* посредством запуска сценария *~/bin/proj1.sh*;
- 5) отразить результаты работы в личном отчёте.

2.1.4 Реализация проекта *proj1* в инструментальной среде IDE Eclipse EE

Eclipse EE является основным интегрированным инструментальным средством (*IDE*, *Integrated Development Environment*) языка Java используемым для разработки программного обеспечения (*ПО*) в изучаемой дисциплине.

Учебная цель данного пункта работы — первоначальное практическое освоение IDE Eclipse EE по освоению технологии создания классических проектов на языке Java.

Практическая цель данного пункта работы — реализация учебного примера приложения *Example1* в форме проекта с именем *proj1* в среде IDE Eclipse EE, а также сравнение результата этой реализации с аналогичным решением полученным при использовании технологии командной строки.

Реализацию поставленных целей проведём в виде последовательности выполнения следующих этапов работ:

- 1) запуск среды разработки Eclipse EE;
- 2) открытие проекта с именем *proj1*;
- 3) создание класса приложения с именем *Example1* и *package ru.tusur.asu*;
- 4) запуск проекта *proj1* на исполнение и изучение инфраструктуры размещения файлов проекта;
- 5) создание запускаемого архива проекта *proj1* в виде файла *~/lib/proj1.1.jar*;
- 6) сравнительное исследование структуры и функционирования архивов *~/lib/proj1.jar* и *~/lib/proj1.1.jar*.

Запуск среды разработки Eclipse EE.

Подготовку и запуск среды разработки Eclipse EE следует произвести в соответствии с методическими указаниями пункта 1.2.4, изложенными в тексте лабораторной работы №2 данного пособия.

Открытие проекта с именем proj1.

Открыв значком «EclipseEE» среду разработки и закрыв вкладку «Welcome», мы переходим к процессу создания первого проекта с именем **proj1**.

В главном меню запущенной IDE, выберем: **File**→**New**. И в появившемся окне, как показано на рисунке 2.16, выберем тип проекта «**Java Project**» и, активировав кнопку «**Next >**», переходим к окну задания параметров проекта, показанному на рисунке 2.17.

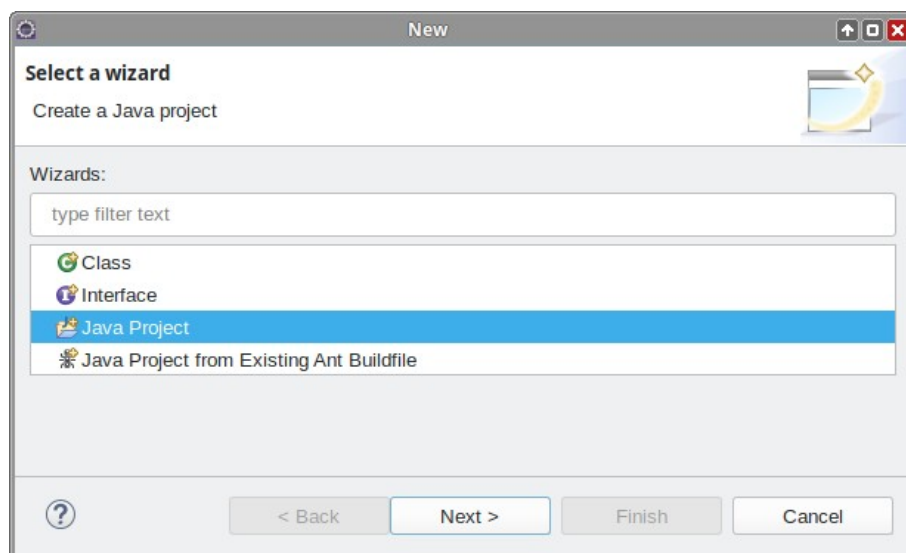


Рисунок 2.16 — Выбор типа проекта «Java Project»

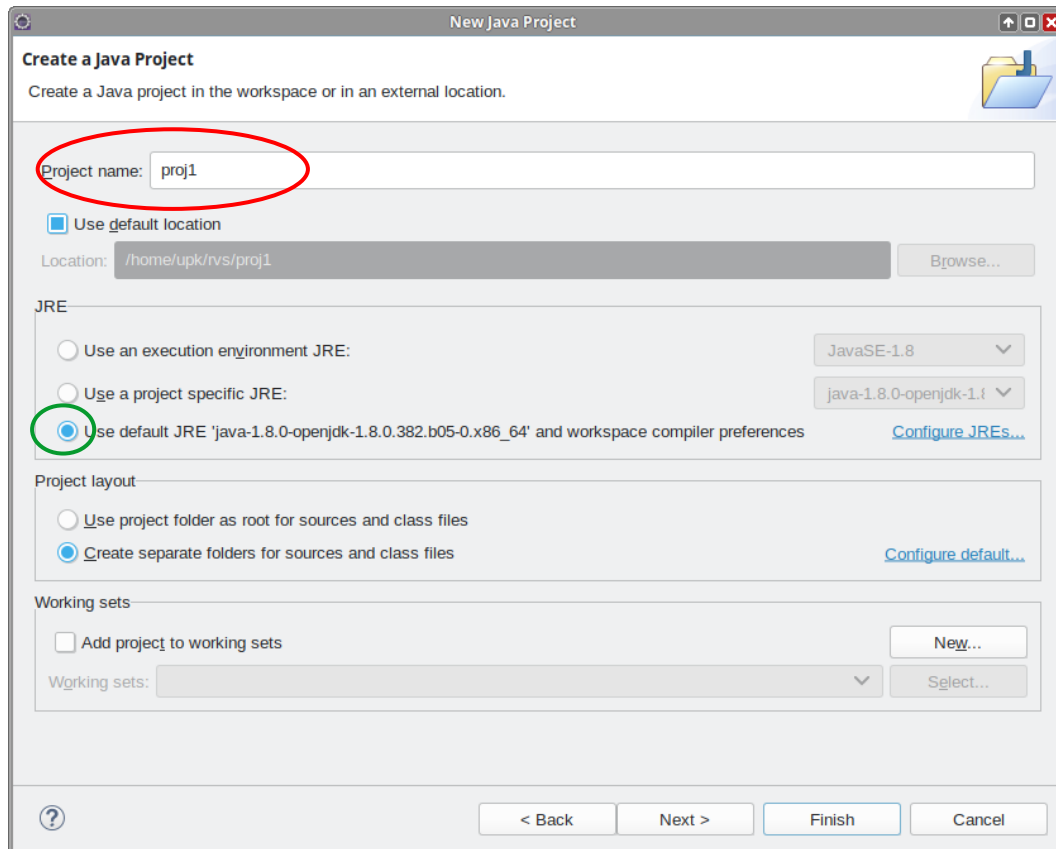


Рисунок 2.17 — Задание имени проекта и установка нужного типа JRE

Задав имя проекта (выделено красным цветом) и желаемый тип (выделено зелёным цветом), нажимаем кнопку «**Finish**», завершая создание проекта с именем **proj1**. В частности может появиться окно показанное на рисунке 2.18 и предлагающее открыть перспективу для вывода результатов запуска проекта. Следует согласиться с этим предложением, нажав кнопку «**Open Perspective**».



Рисунок 2.18 — Предложение открыть перспективу для создаваемого проекта

В результате указанных выше действий на ша система разработки перейдёт в состояние, показанное на рисунке 2.19.

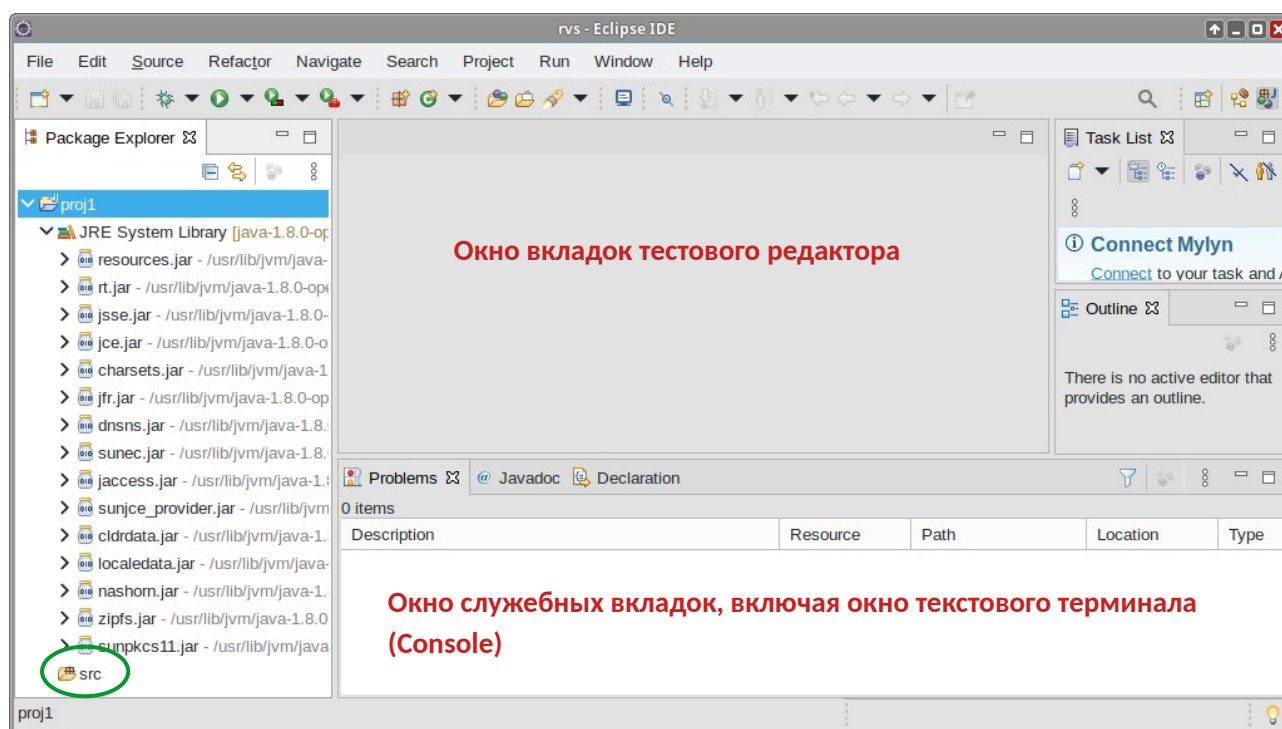


Рисунок 2.19 — Основное окно Eclipse IDE с единственным открытым проектом proj1

Основное окно Eclipse IDE (для проектов типа Java SE) имеет следующие три внутренних окна:

- 1) окно «*Package Explorer*» содержит список всех открытых в системе проектов; в нашем случае открыт только один проект с именем **proj1**, который имеет внутренний список подключённых к проекту библиотек «*JRE System Library*», и список созданных в этом проекте классов «*src*», который на данный момент является пустым;

- 2) окно вкладок тестового редактора, расположенное в верхней средней части основного окна;
- 3) окно служебных вкладок, включая окно «*Console*», которое будет появляться во время запуска приложений, использующих каналы стандартного ввода/вывода.

Создание класса приложения с именем **Example1** и package **ru.tusur.asu**.

Чтобы создать приложение с именем **Example1**, необходимо в окне «*Package Explorer*» выделить мышкой значок «*src*», обведённый зелёным эллипсом, и правой кнопкой мыши активировать контекстное меню.

В контекстном меню следует мышкой выбрать **New/Class**, после чего появится диалоговое окно «*New Java Class*», которое необходимо отредактировать, как это показано на рисунке 2.20 (редактируемые поля помечены красной звёздочкой).

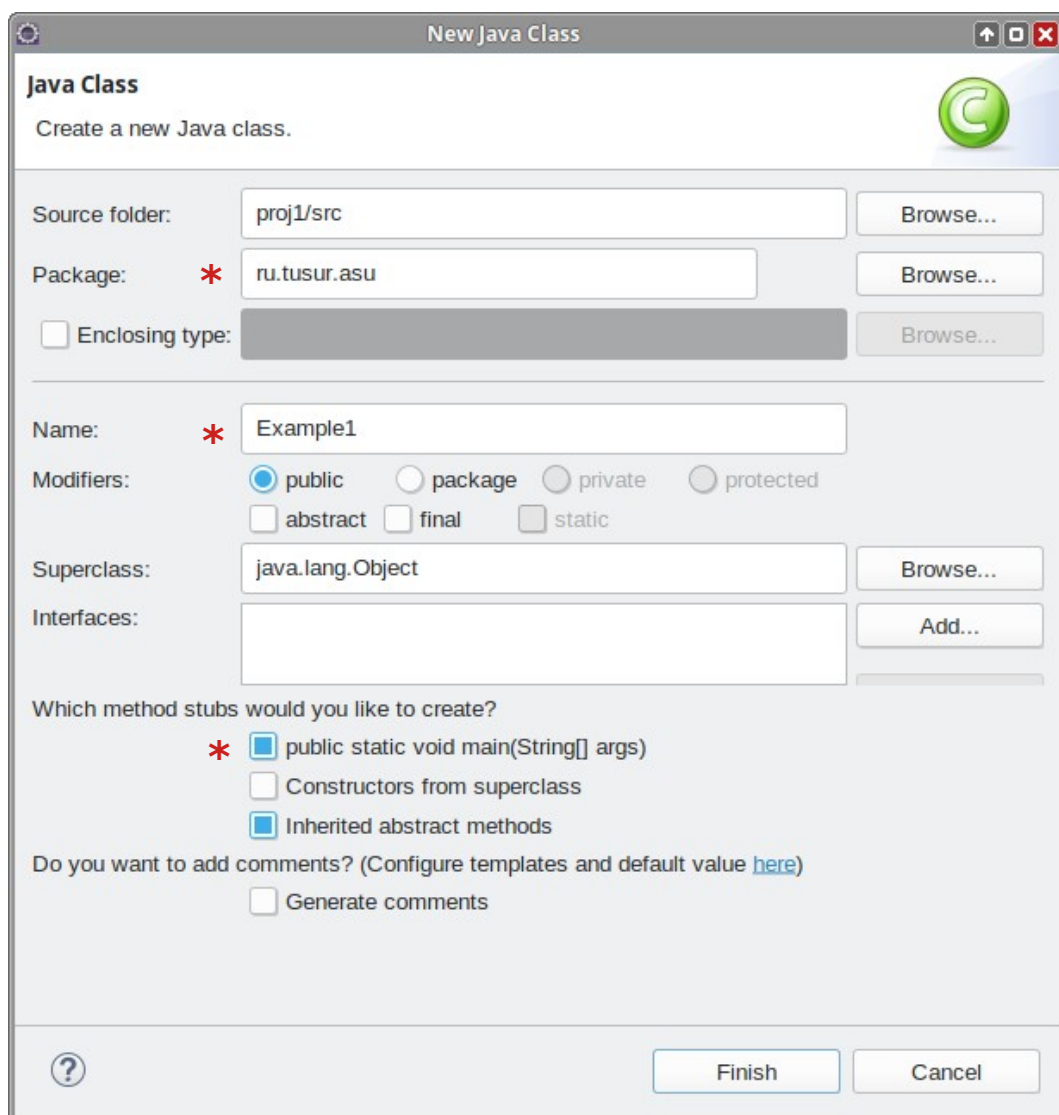


Рисунок 2.20 — Диалоговое окно создания класса проекта proj1

Активируя кнопку «*Finish*», мы создаём шаблон класса **Example1**, как это показано на рисунке 2.21. Соответствующий этому классу файл **Example1.java** выводится в в окно редактирования среды разработки. Стрелкой показан значок запуска приложения на исполнение.

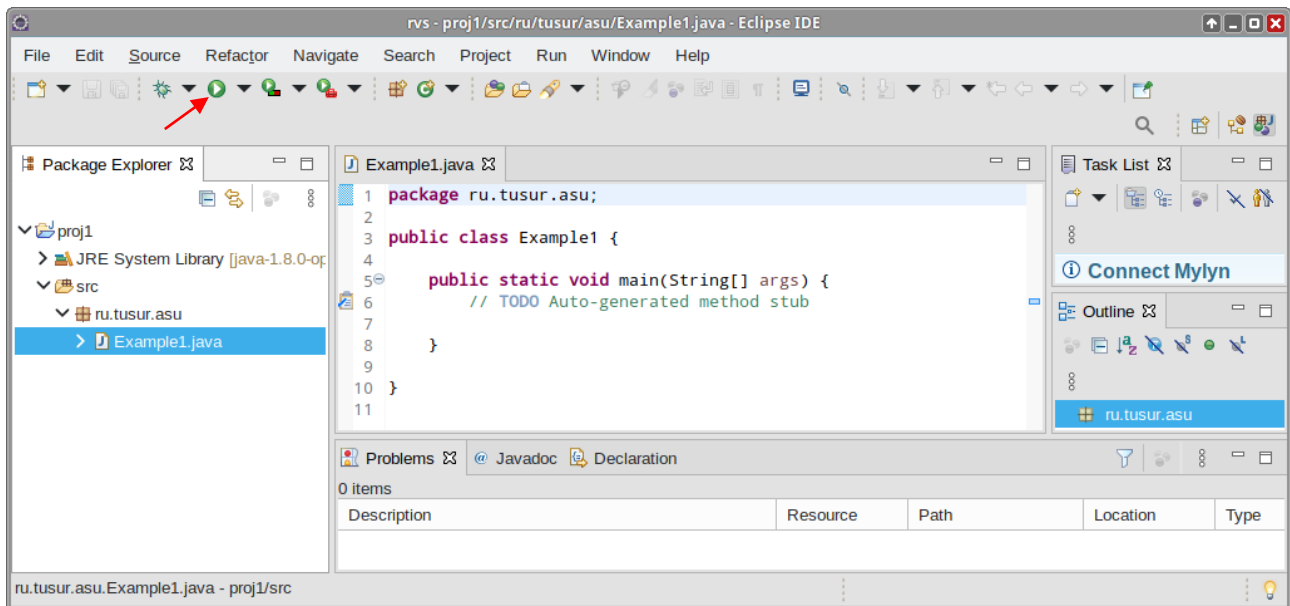


Рисунок 2.21 — Окно Eclipse IDE с открытым на редактирование шаблоном класса Example1

Запуск проекта proj1 на исполнение и изучение инфраструктуры размещения файлов проекта.

Для запуска целевого приложения *Example1* в среде проекта *proj1* необходимо:

- 1) открыть файл `~/src/rvs/Example1.java` редактором *mousepad* и заменить им текст шаблона во вкладке «*Example1.java*» системы Eclipse IDE;
- 2) запустить приложение на исполнения с помощью кнопки, указанной красной стрелкой на рисунке 2.21.

Результат запуска приложения *Example1* показан на рисунке 2.22.

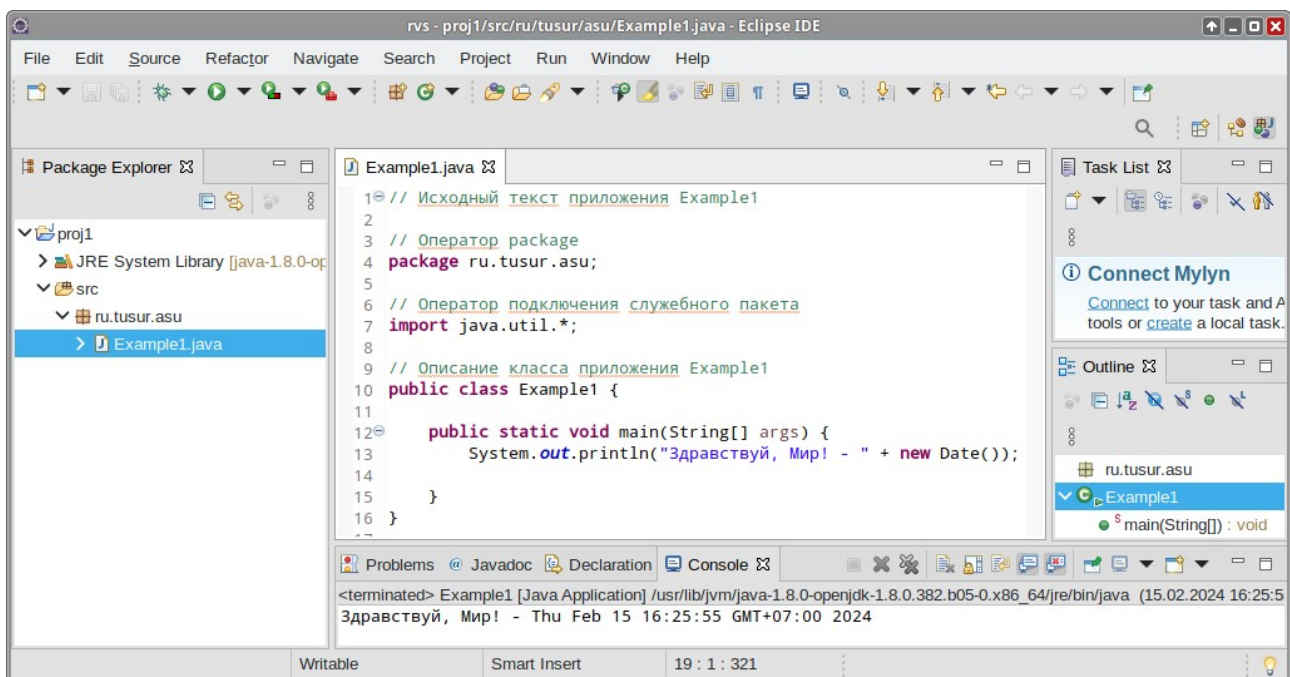


Рисунок 2.22 — Результат запуска приложения Example1 в среде IDE Eclipse

Примечание — В одном проекте может быть создано множество классов предназначенных для запуска в среде исполнения Eclipse IDE, поэтому следует учитывать следующие правила: на запуск отправляется тот файл класса, который выделен в редакторе среды разработки; если содержимое файла было отредактировано, но не сохранено, то имя вкладки отмечается символом «звездочка»; сохранение исходного текста файла осуществляется нажатием клавиш «**Ctrl-S**»; после сохранения файла осуществляется его автоматическая компиляция и в тексте файла помечаются места, содержащие синтаксические ошибки.

Убедившись в работоспособности проекта **proj1** следует изучить его файловую инфраструктуру. Для этого лучше использовать файловый менеджер «*Midnight Commander*», как это показано на рисунке 2.23.

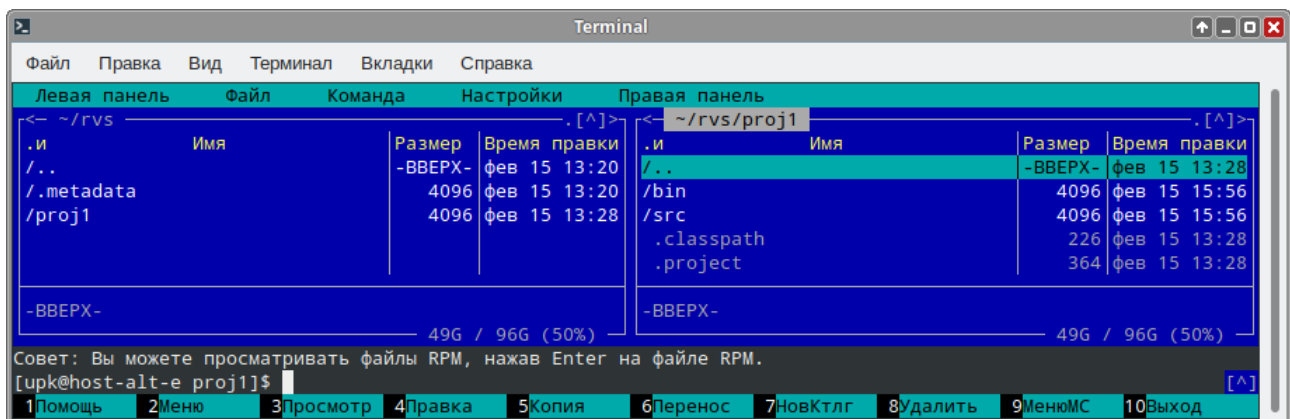


Рисунок 2.23 — Начальный этап исследования структуры ФС проекта proj1

Левая панель файлового менеджера **mc** показывает общую структуру каталога проектов **~/rvs**. Хорошо видно, что Eclipse IDE создал всего лишь один проект с именем **proj1**.

Правая панель файлового менеджера **mc** показывает содержимое самого каталога проекта **proj1**.

Студенту следует самостоятельно исследовать содержимое каталогов **bin** и **src**, а полученный результат отразить в личном отчёте.

Создание запускаемого архива проекта proj1 в виде файла ~/lib/proj1.1.jar.

Выделим мышкой в окне «*Package Explorer*» значок **proj1** и правой кнопкой мышки активируем контекстное меню. В контекстном меню выберем пункт «*Export...*», результате чего появится окно диалога «*Export*» показанное на рисунке 2.24.

В окне «*Export*» выберем тип создаваемого архива «*Runnable JAR file*» и активируем кнопку «*Next >*». В результате появится окно диалога «*Runnable JAR File Export*» показанное на рисунке 2.25.

Обратите внимание, что в самом простейшем случае, в окне диалога на рисунке 2.25 необходимо заполнить два поля:

- 1) *Launch configuration:* - задаёт как имя архивируемого проекта (**proj1**), так и имя класса, который будет исполняться (**Example1**);
- 2) *Export destination:* - задаёт полный путь файла архива; мы используем для архива имя **proj1.1.jar**, чтобы не перезаписывать архив созданный в командной строке.

После активации кнопки «*Finish*» целевой архив будет создан и помещён в нужное место файловой системы ОС.

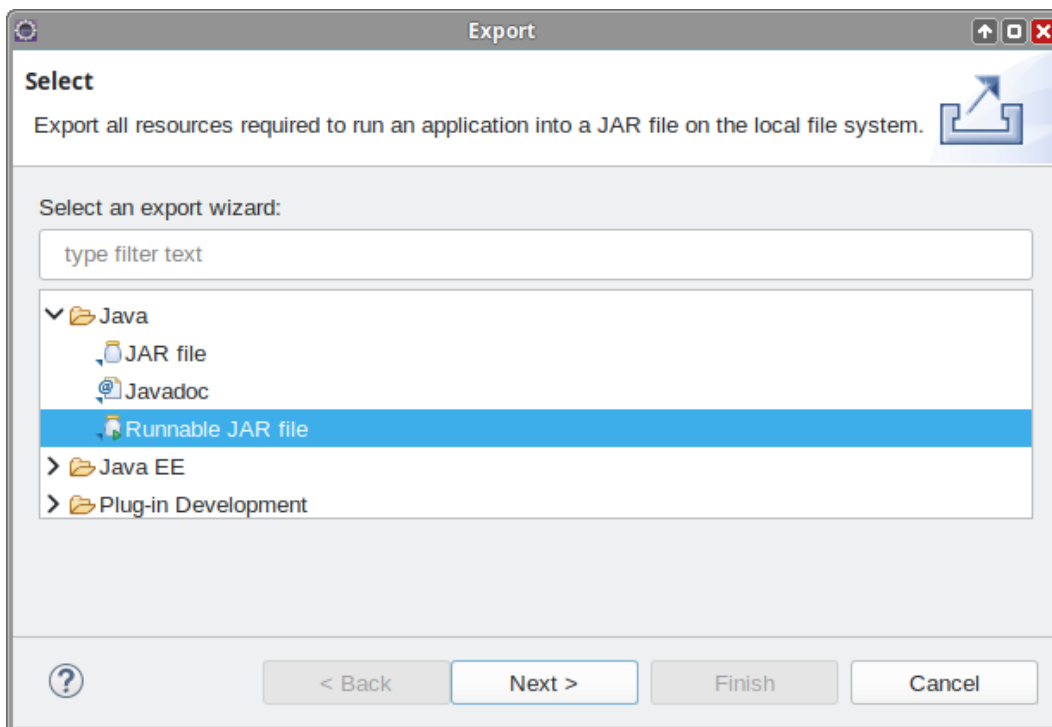


Рисунок 2.24 — Выбор типа создаваемого архива JAR

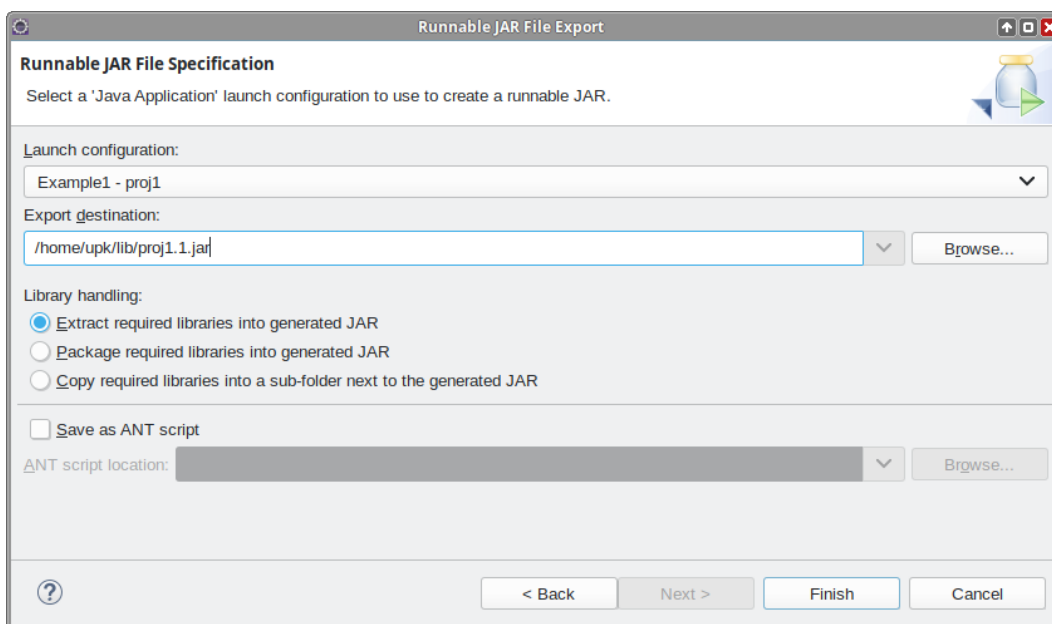


Рисунок 2.25 — Выбор архивируемого проекта и задание полного пути создаваемого архива

Сравнительное исследование структуры и функционирования архивов `~/lib/proj1.jar` и `~/lib/proj1.1.jar`.

Прежде чем проводить сравнительное исследование созданных архивов следует перейти в каталог `~/lib` и убедиться, что архив *proj1.1.jar* создан. Указанное действие показано на рисунке 2.26.

Затем следует проверить работоспособность вновь созданного архива, например, командой подобной выражению (2.6).

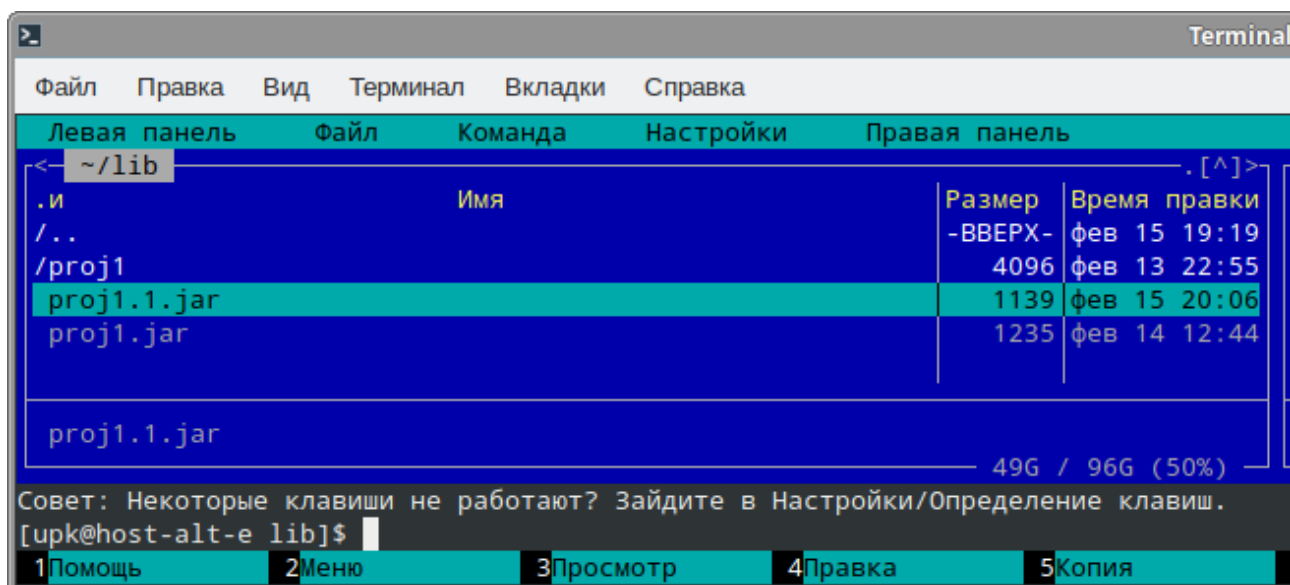


Рисунок 2.26 — Проверка наличия архивов в каталоге ~/lib

Сравнительный анализ созданных архивов *proj1.jar* и *proj1.1.jar* студенту следует провести самостоятельно в пределах третьего учебного задания.

Третье учебное задание:

- 1) выполнить работы по созданию и запуску на исполнение проекта *proj1* в инструментальной среде Eclipse IDE;
- 2) выполнить исследование файловой структуры проекта *proj1*, начиная с содержимого его каталога *~/rvs/proj1*;
- 3) выполнить в инструментальной среде Eclipse IDE создание архива *~/lib/proj1.1.jar*;
- 4) провести сравнительный анализ созданных архивов *proj1.jar* и *proj1.1.jar*;
- 5) отразить результаты работы в личном отчёте;
- 6) завершить выполнение третьей лабораторной работы.

2.2 Лабораторная работа №4.

Классы, интерфейсы и методы языка Java

Предыдущая лабораторная работа была посвящена технологиям манипулирования с мета-объектами языка Java, к которым относятся исходные тексты программ и файлы результатов компиляции исходных текстов, создание пакетов программ и их архивное представление, использование утилит командной строки *java*, *javac*, *jar* и интегрированного инструментального средства разработки проектов — *Eclipse IDE*.

Настоящая лабораторная работа посвящена синтаксическим конструкциям самого языка Java, которые концентрируются в понятиях **классов**, **интерфейсов** и **методов**.

Учебная цель данной работы — изучение технологии создания и использования базовых синтаксических конструкций языка Java, ограниченных теоретическим материалом подразделов 2.2 и 2.3 учебного пособия [1].

Теоретическая часть работы охватывает:

- 1) изучение общей структуры класса приложения, включающая использование методов его *запуска*, *завершения работы*, а также стандартных средств ввода/вывода;
- 2) изучение управляющих операторов языка Java и сопутствующих им классов демонстрируются конкретными примерами обязательными примерами приложений и учебным справочным материалом полного руководства [3];
- 3) изучение простых типов данных и соответствующих им классов-обёрток языка Java соответствует содержанию таблицы 2.3.

Таблица 2.3 — Изучаемые типы данных и классы-обёртки языка Java версии 8

Тип/класс	Длина (байт)	Описание типа/класса
<i>boolean</i>	Не определено	Логические значения: <i>true/false</i>
<i>Boolean</i>	-	Класс-обёртка простого типа <i>boolean</i>
<i>byte</i>	1	Целое знаковое число: <i>-128..127</i>
<i>Byte</i>	-	Класс-обёртка простого типа <i>byte</i>
<i>char</i>	2	Целое без знаковое число: <i>0..65535</i>
<i>Character</i>	-	Класс-обёртка простого типа <i>char</i>
<i>short</i>	2	Целое знаковое число: <i>-32768..32767</i>
<i>Short</i>	-	Класс-обёртка простого типа <i>short</i>
<i>int</i>	4	Целое знаковое число: <i>-2147483648.. 2147483647</i>
<i>Integer</i>	-	Класс-обёртка простого типа <i>int</i>
<i>long</i>	8	Вещественное знаковое число: <i>-9.2*10¹⁸..9.2*10¹⁸</i>
<i>Long</i>	-	Класс-обёртка простого типа <i>long</i>
<i>float</i>	4	Вещественное знаковое число: <i>-3.4*10³⁸..3.4*10³⁸</i>
<i>Float</i>	-	Класс-обёртка простого типа <i>float</i>
<i>double</i>	8	Вещественное знаковое число: <i>-1.8*10³⁰⁸..1.8*10³⁰⁸</i>
<i>Double</i>	-	Класс-обёртка простого типа <i>double</i>

Практическая цель работы — практическое закрепление проектной работы с синтаксическими конструкциями *классов*, *интерфейсов* и *методов* в интегрированной среде разработки Eclipse IDE, охватывающих наиболее общие навыки программирования на языке Java.

Общее учебное задание — *реализовать и исследовать* в проекте **proj2** интегрированной среды разработки Eclipse IDE набор учебных примеров, обеспечивающих достижение учебной и практической целей лабораторной работы №4. Выполняемые операции и действия работы изложить в личном отчёте, в виде последовательности следующих пунктов:

- 1) классы, объекты и объектные переменные языка Java;
- 2) объекты типа **String** и стандартный вывод результатов работы программ;
- 3) преобразования простых типов данных;
- 4) использование массивов информации простых и объектных типов данных;
- 5) работа со строками данных типов **String** и **StringBuffer**.

2.2.1 Классы, объекты и объектные переменные языка Java.

Мы уже знаем, что язык Java имеет пакетную организацию. Все его базовые конструкции определены в пакете **java.lang** и поддерживаются виртуальной машиной JVM, которая представлена в среде ОС утилитой **java**.

Основные синтаксические конструкции языка Java опираются на определения:

- 1) **класса** — *обязательной* синтаксической конструкции любой программы написанной на Java, неявным супер-классом которой является конструкция **public class Object**;
- 2) **метода** — *необязательной* синтаксической конструкции любого класса программы, написанной на языке Java, которая ассоциируется с функционалом класса.

Указанные синтаксические конструкции хорошо согласуются с соответствующими теоретическими понятиями языков объектно ориентированного программирования (ООП) и хорошо должны быть известны студентам, уже изучившим языки C/C++.

Поскольку данная и все последующие лабораторные работы используют интегрированную среду разработки Eclipse IDE, то обязательными дополнительными понятиями являются конструкции **проекта** и **package**, с которыми студенты уже познакомились в процессе выполнения предыдущей лабораторной работы.

Первое учебное задание:

- 1) запустить среду разработки Eclipse IDE;
- 2) создать проект с именем **proj2**;
- 3) в проекте **proj2** создать класс с именем **Main2**, входящий в проект **rvs.pr1**;
- 4) заменить содержимое класса **Main2** на содержимое текста листинга 2.1.

Листинг 2.1 — Исходный текст класса Main2 проекта proj2

```
package rvs.pr1;

public class Main2 {
    /**
     * Глобальные переменные.
     * @param args
```

```

    */
    private      String version = "1.0";
    public static String title   = "Пример программы";

    public String getVersion()
    {
        return version;
    }

    public static void main(String[] args) {
        /**
         * Создание локального объекта.
         */
        Main2 m2 = new Main2();
        System.out.println(title + ": " + m2.getVersion());
    }
}

```

Примечание — В целях экономии времени набора исходных текстов программ их файлы размещены в каталоге «~/Документы/Исходные тексты проектов».

Представленный исходный текст класса **Main2** демонстрирует нам:

- 1) задание комментариев класса, способных включать в себя аннотации, позволяющие автоматически создавать документацию реализуемых проектов;
- 2) определение двух (глобальных в пределах класса) объектных переменных **version** и **title** типа **String**, которые имеют разную «видимость» вне класса и «уникальность» для создаваемых объектов;
- 3) определение двух методов **getVersion()** и **main(...)**, первый из которых возвращает значение типа **String**, а второй метод не возвращает ничего, поскольку имеет тип **void**.

Обратим внимание на особенности задания и использования метода **main(...)**, который обеспечивает возможность запуска на исполнение и основной алгоритм функционирования класса **Main2**:

- 1) метод *должен иметь* спецификатор доступа **public**, иначе он не позволит запустить класс на исполнение;
- 2) *должен иметь* модификатор **static**, иначе он не будет видим системе запуска приложения;
- 3) *имеет единственный аргумент* в виде списка элементов типа **String**, позволяющий прочитать аргументы запускаемого приложения;
- 4) *не возвращает значения*, поскольку имеет тип **void**.

Примечание — На самом деле, при нормальном завершении метода **main(...)**, в ОС возвращается целочисленное значение **ноль**.

Чтобы приложение языка Java возвращало ненулевое целочисленное значение, следует его завершать методом задаваемым выражением (2.9).

System.exit(целочисленный код возврата); **(2.9)**

Особенности имеются и при использовании других методов внутри **main(...)**.

Например, чтобы в методе *main(...)* использовать другие методы этого же класса необходимо сначала использовать конструктор этого класса для создания объекта. В частности, если описываемый класс не имеет супер-класса, а его конструктор не подразумевает наличия аргументов, то такой конструктор можно не описывать.

Примечание — В отличие от языка C++, в языке Java имеются только динамически создаваемые объекты и все переменные языка реально являются объектными ссылками. Более того в языке Java отсутствует и само понятие ссылки и соответствующие им операции адресации, доступные в языках C/C++.

Указанное примечание является очень важным для понимания «механизмов» работы языка Java, поэтому студенту следует выполнить и описать выполнение следующего задания.

Второе учебное задание:

- 1) создать в проекте *proj2* класс с именем *Example2*, согласно текста листинга 2.2;
- 2) создать в проекте *proj2* класс с именем *Example3*, согласно текста листинга 2.3;
- 3) провести совместное тестирование работы созданных классов;
- 4) результаты исследования отразить в личном отчёте.

Листинг 2.2 — Исходный текст класса *Example2* проекта *proj2*

```
package ru.tusur.asu;

public class Example2 {

    // Объявление переменной типв String
    String text1;

    // Объявление статического массива из двух целых чисел
    static int[] im = new int[2];

    // Конструктор класса
    Example2(String text2, int n) {
        // Присваиваем значение части переменных
        text1 = text2;
        im[0] = n;
    }

    // Первый (обычный) метод
    public void print1() {
        System.out.println(text1 + ":"
            + " im[0]=" + im[0]
            + " im[1]=" + im[1]);
    }

    // Второй (статический) метод
    public static void print2() {
        System.out.println("Статический метод:"
            + " im[0]=" + im[0]
            + " im[1]=" + im[1]);
    }
}
```

Обратите внимание, что класс *Example2* не содержит метода *main(...)*.

Листинг 2.3 — Исходный текст класса *Example3* проекта *proj2*

```
package ru.tusur.asu;

public class Example3 {

    public static void main(String[] args) {

        // Объявляем и создаём объект obj1 класса Example2
        Example2 obj1 =
            new Example2("Первый вызов обычного метода", 1);

        // Вызываем обычный метод объекта obj1
        obj1.print1();

        // Вызываем статический метод класса Example2
        Example2.print2();

        // Объявляем и инициализируем объект obj2 класса Example2
        Example2 obj2 = obj1;

        // Вызываем обычный метод объекта obj2
        obj2.print1();

        // Пересоздаём объект obj1 класса Example2
        obj1 =
            new Example2("Второй вызов обычного метода", 2);

        // Вызываем обычный метод объекта obj1
        obj1.print1();

        // Вызываем статический метод класса Example2
        Example2.print2();

        // Удаляем объект для obj1 и obj2: "сборка мусора"
        obj1 = null;
        obj2 = null;

    }
}
```

Примечание — При выполнении второго учебного задания рекомендуется воспользоваться текстом пояснений, которые приведены в пункте 2.2.5 учебного пособия [1].

2.2.2 Объекты типа *String* и стандартный вывод информации

Уже рассмотренные примеры исходных текстов классов языка Java показывают широкое использование строковых объектов типа *String*. Кроме того эти строковые объекты постоянно используются для вывода информации на текстовый терминал (*Console*), как в среде разработки Eclipse IDE, так и в командной строке терминала.

Учебная цель данного пункта работы — изучение основ форматирования строковых объектов типа *String*, а также обеспечение печати этих объектов на стандартных каналах вывода информации.

В общем случае для вывода *текстовых программных сообщений* языка Java предусмотрено два уже открытых канала вывода, которые представлены двумя объектами:

- 1) ***java.lang.System.out*** — для вывода нормальных сообщений, например, результатов расчётов программы;
- 2) ***java.lang.System.err*** — для вывода ошибок.

Оба этих объекта имеют тип ***PrintStream***, принадлежат пакету ***java.lang*** и имеют множество методов, обеспечивающих преобразование различных типов данных для вывода их на печать. В простейших случаях вывода текста обычно используются два метода:

- 1) ***println(String str)*** — печатает содержимое строки ***str*** и в конце *добавляет символ* перевода строки;
- 2) ***print(String str)*** — печатает содержимое строки ***str***, но в конце *не добавляет символа* перевода строки.

Примечание — Более подробно классы и методы ввода/вывода осваиваются в следующей лабораторной работе.

Для форматированного стандартного вывода используется метод заданный общим выражением (2.10).

printf(String format, Object ... args); **(2.10)**

где:

- ***format*** — строка формата, общий синтаксис которой определён выражением (2.11);
- ***args*** — аргументы, на которые ссылаются спецификаторы формата в строке формата; если аргументов больше, чем спецификаторов формата, дополнительные аргументы игнорируются; количество аргументов является переменным и может быть нулевым.

%[argument_index\$] [flags] [width] [.precision] conversion **(2.11)**

Синтаксис строки формата (2.11) аналогичен синтаксису языков C/C++ и содержит следующую семантику:

- а) необязательный элемент ***arguments_index*** — десятичное целое число, обозначающее позицию аргумента в списке аргументов; на первый аргумент ссылаются «***1\$***», на второй — «***2\$***» и так далее;
- б) необязательный элемент ***flags*** — это набор символов, которые изменяют формат вывода; набор допустимых флагов зависит от символа преобразования (см. далее);
- в) необязательный элемент ***width*** — ширина вывода, представляющее собой неотрицательное десятичное целое число и указывающее минимальное количество символов, которое будет записано в вывод;
- г) необязательная часть ***precision*** (точность) — неотрицательное десятичное целое число, обычно используемое для ограничения количества десятичных символов вещественных чисел; конкретное поведение *зависит от требуемого преобразования*;
- д) требуемое преобразование (***conversion***) — *символ преобразования*, указывающий, как аргумент должен быть отформатирован; набор допустимых преобразований для данного аргумента зависит от типа данных используемого аргумента.

Символы преобразования формата (2.11) — следующие:

- 1) ***b*** — если аргумент ***arg*** равен нулю, то результатом будет "***false***"; если ***arg*** имеет логическое значение, то результатом вывода является строка, которая возвращается функ-

- цией ***String.valueOf()***; в противном случае результат будет «***true***»;
- 2) ***h*** — если аргумент ***arg*** равен нулю, то результат равен нулю; в противном случае результат получается путём вызова метода ***Integer.toHexString()***;
 - 3) ***s*** — если аргумент ***arg*** равен нулю, то результат будет равен нулю; Если ***arg*** не равен пустой строке, то результат получается вызовом метода ***arg.toString()***;
 - 4) ***c*** — результатом является символ Unicode;
 - 5) ***d*** — результат форматируется как десятичное целое число;
 - 6) ***o*** — результат форматируется как восьмеричное целое;
 - 7) ***x*** — результат форматируется как шестнадцатеричное целое;
 - 8) ***e*** — результат форматируется как десятичное число в компьютеризированной научной нотации;
 - 9) ***f*** — результат форматируется как десятичное число с плавающей точкой;
 - 10) ***g*** — результат форматируется с использованием компьютеризированной научной нотации или десятичного формата, которые зависят от точности и значения самого числа после его округления;
 - 11) ***a*** — результат форматируется как шестнадцатеричное число с плавающей запятой со значениями и показателем степени;
 - 12) ***t*** — префикс даты/времени для символов преобразования даты и времени;
 - 13) ***%*** - результатом является литерал ***"%"*** (***"\u0025"***);
 - 14) ***n*** — разделитель строк; результатом является специфичный для платформы разделитель строк.

Примечание — Если в программе необходимо получить строку некоторого заданного формата, то можно воспользоваться статическим методом ***format(...)*** объекта класса ***String***, как это показано выражением (2/12).

String str = String.format(String ss, Object ... args); (2.12)

Закрепим изложенный учебный материал отдельным учебным заданием.

Третье учебное задание:

- 1) создать в проекте ***proj2*** класс с именем ***RvsOut***, согласно текста листинга 2.4;
- 2) провести разумную модификацию и тестирование работы созданного класса;
- 3) результаты исследования отразить в личном отчёте.

Листинг 2.4 — Исходный текст класса ***RvsOut*** проекта ***proj2***

```
package rvs.pr1;

import java.io.PrintStream;

public class RvsOut {

    public static void main(String[] args) {
        /**
         * Определение и задание начальных значений.
         */
        int ix = 10;
```

```

float fy = 111;
double dy = 45.78;

/**
 * Вывод на печать.
 */
PrintStream msg = System.out;

msg.printf("Вывод целого числа ix = %1$d; %1$o \n", ix);
msg.printf("Вывод float числа fy = %1$f; %1$g \n", fy);
msg.printf("Вывод double числа dy = %1$f; %1$e \n", dy);
}
}

```

2.2.3 Преобразования простых типов данных

Язык Java является языком объектно-ориентированного программирования (ООП), поэтому его семантика опирается на три базовых понятия: *класс*, *метод класса* и *объект класса*:

- 1) **класс** — именованный *объектный тип данных*, основанный на свойствах базового супер-класса *java.lang.Object*;
- 2) **метод класса** — именованное *свойство класса*, обеспечивающее функциональность его объектов;
- 3) **объект класса** — именованный и актуализированный посредством метода конструктора класса — *элемент класса*, способный реально функционировать в программной среде ОС (в среде приложения Java под управлением JVM).

Кроме общей модели объектного типа данных язык Java поддерживает восемь *простых типов данных* и их *классы-оболочки*, которые были ранее представлены в [таблице 2.3](#):

- 1) **простой тип данных** — специальный тип данных, который не основан на свойствах супер-класса *java.lang.Object*, не имеет методов, но имеет ассоциативную связь с *логическим, числовым или символьным* типами данных; функциональность объектов простых типов данных обеспечивается *операторами* самого языка Java;
- 2) **класс-оболочка** — класс *объектного типа данных*, соответствующий конкретному *простому типу данных* и имеющий методы взаимного их преобразования.

Учебная цель данного пункта — изучение правил преобразования простых типов данных в объектные типы — обратно.

Примечание — По правилам языка Java, если какой-либо класс имеет метод с модификатором **static**, то такой метод может быть вызван по *имени класса*, после которого ставится *точка*, а затем указывается сам *метод* (см. описание модификаторов методов в пункте 2.2.3 учебного пособия [1]).

Преобразование типов гораздо легче продемонстрировать конкретным примером, чем излагать эти правила в текстовой форме.

Четвёртое учебное задание:

- 1) создать в проекте *proj2* класс с именем *Types*, согласно текста листинга 2.5;
- 2) провести разумную модификацию и тестирование работы созданного класса;
- 3) результаты исследования отразить в личном отчёте.

Листинг 2.5 — Исходный текст класса `Types` проекта `proj2`

```
package rvs.pr2;

/**
 * Примеры преобразования типов.
 * @author vgr
 */
public class Types {

    public static void main(String[] args) {

        /**
         * Преобразование целого числа в объект и другие типы.
         */
        int i = 5;
        Integer ii = Integer.valueOf(i);
        float f = ii.floatValue();
        double d = ii.doubleValue();
        long l = ii.longValue();

        // Печать объявленных и преобразованных значений
        System.out.println(" i = " + i
            + "; ii = " + ii
            + "; f = " + f
            + "; d = " + d
            + "; l = " + l);

        /**
         * Преобразование целого числа в строку.
         */
        String ss = String.valueOf(i);
        /**
         * Преобразование строки во float и другие типы.
         */
        Float ff = Float.valueOf(ss);
        f = ff.floatValue();
        d = ff.doubleValue();
        l = ff.longValue();

        // Печать объявленных и преобразованных значений
        System.out.println("ss = " + ss
            + "; ff = " + ff
            + "; f = " + f
            + "; d = " + d
            + "; l = " + l);

    }
}
```

При выполнении четвёртого задания следует обратить внимание, что:

- 1) все классы оболочки имеют статические методы `valueOf(...)`, которые преобразуют простой тип класса или объект типа ***String*** в объект искомого класса;
- 2) все объекты классов оболочек имеют множество методов, например, `longValue()`, преобразующие объект этого класса в объект одного из простых типов данных;
- 3) если в редакторе исходного текста среды Eclipse IDE после имени класса или объекта класса поставить точку, то среда разработки выдаёт контекстное меню со всеми допустимыми для использования методами.

2.2.4 Использование массивов простых и объектных типов данных

В языке Java, как и в других языках программирования, широко используется понятие *массива*, которое одинаково интерпретируются как для *простых*, так и для *объектных типов данных*.

Массив — *объектный тип данных*, определённый на отдельном простом или объектном типе данных и задающий его *упорядоченную конечную мерность*, определяющую систему координат объединяемой массивом группы объявленных типов данных.

Например, выражение (2.13) определяет объектную переменную *aa*, в виде одномерного массива типа *int*, выражение (2.14) — объектную переменную *bb*, в виде двумерного массива типа *String*.

```
int[] aa; (2.13)
```

```
String[] [] bb; (2.14)
```

Примечание — В языке Java массивы являются объектами, которые могут создаваться и передаваться в другие методы. Перед использованием массивов для них сначала нужно определить размерность каждой *мерности*, а затем — выделить требуемый объем памяти.

В целом процесс создания и использования массивов Java можно разделить на три основных этапа:

- 1) объявление массива;
- 2) выделение памяти для элементов массива;
- 3) инициализация элементов массива.

Как и в предыдущем пункте лабораторной работы, здесь мы ограничим процесс обучения демонстрацией конкретного примера, поясняющего суть вопроса. Для более глубокого изучения данного вопроса следует изучить материал главы 3 рекомендуемого источника [3].

Пятое учебное задание:

- 1) создать в проекте *proj2* класс с именем *Array2*, согласно текста листинга 2.6;
- 2) создать в проекте *proj2* класс с именем *Array3*, согласно текста листинга 2.7;
- 3) провести разумную модификацию и тестирование работы созданных классов;
- 4) результаты исследования отразить в личном отчёте.

Листинг 2.6 — Исходный текст класса *Array2* проекта *proj2*

```
package rvs.pr3;
/**
 * Примеры определения массивов.
 * @author vgr
 */
public class Array2 {

    public static void main(String[] args) {
        int max = 20;
        /**
         * Объявление массивов:
         */
```

```

double[] d1;
int[] i1;
long[][] l1;
String[] s1;
/**
 * Выделение памяти для элементов массива:
 */
d1 = new double[40];
i1 = new int[max];
l1 = new long[10][max];
s1 = new String[10];
/**
 * Инициализация элементов массива:
 */
for(int i=0; i<2*max; i++)
    d1[i] = 2.5 * i;

for(int i=0; i<10; i++)
{
    i1[i] = 30 * i;
    for(int j=0; j<max; j++)
        l1[i][j] = i + j;
}
s1[0] = "Текст0";
s1[1] = "Текст1";
/**
 * Одновременное и инициализация массивов
 * в виде констант
 */
double[] d2 = {2.5, 4.7, 5.6}; // Три элемента
System.out.println("d2 = " + d2[0] + ", " + d2[1] + ", " + d2[2]);

long[][] l2 = {{7, 3, 2},
               {6, 5, 20}}; // Массив 2x3
System.out.println("l2[0] = " + l2[0][0] + ", " + l2[0][1]
                  + ", " + l2[0][2]);
System.out.println("l2[1] = " + l2[1][0] + ", " + l2[1][1]
                  + ", " + l2[1][2]);

String[] s2 = {"Текст0",
               "Текст1",
               "Текст2"}; // Три элемента
System.out.println("s2 = " + s2[0] + ", " + s2[1] + ", " + s2[2]);

}
}

```

Обратите внимание, что исходный текст класса *Array2*, на листинге 2.6, условно разделён на две части:

- 1) первая часть программы представляет *раздельное* объявление, выделение памяти и инициализацию объектов *d1*, *i1*, *l1* и *s1* разных типов и мерности;
- 2) вторая часть программы представляет вариант задания объектных *констант массивов*, выполняя *одновременные* объявления, выделение памяти и инициализацию элементов объектов *d2*, *i2*, *l2* и *s2* разных типов и мерности.

Второй пример, реализованный в листинге 2.7, демонстрирует нам следующие две особенности исполнения:

- 1) определение класса *Type3* внутри файла *Array3.java*;

- 2) создание и обработку объектов типа *Type3* внутри основного класса *Array3*.

Листинг 2.7 — Исходный текст класса *Array3* проекта *proj2*

```
package rvs.pr3;

/**
 * Класс, выступающий как элемент массива.
 * @author vgr
 */
class Type3
{
    int    i;
    double d;
}

/**
 * Класс использующий другой класс как массив.
 * @author vgr
 */
public class Array3 {

    public static void main(String[] args) {
        /**
         * Объявляем и инициализируем массив объектов.
         */
        int N = 10;
        Type3[] t3 = new Type3[N]; // Выделяем ссылки.
        for(int i=0; i<N; i++)
        {
            t3[i] = new Type3(); // Создаём объект.
            t3[i].i = i;
            t3[i].d = 1.41 * i;
        }
    }
}
```

2.2.5 Работа со строками данных типов *String* и *StringBuffer*

В пункте 2.2.2 уже было отмечено, что программисту постоянно приходится использовать строковые объекты и был рассмотрен пример форматированного преобразования строк и вывода этих строк на стандартный канал вывода *java.lang.System.out*.

В общем случае язык Java предоставляет *два типа строковых объектов*, имеющих не только различные методы работы со строками, но и различные подходы к их созданию и хранению:

- 1) *java.lang.String* — *неизменяемый* строковый объект предполагающий, что все совершаемые со строковым объектом операции приводят к *автоматическому созданию нового строкового объекта* и удалению старого строкового объекта;
- 2) *java.lang.StringBuffer* — *изменяемый* строковый объект, *предполагающего создание в среде JVM динамически изменяемого буфера*, в котором и производятся все изменения целевого строкового объекта.

Учебная цель данного пункта — изучение двух примеров работы со строковыми объектными типами *String* и *StringBuffer*.

Примечание — Для более полного и подробного изучения работы со строковыми объектами рекомендуется прочитать главу 16 источника [3].

Шестое учебное задание:

- 1) создать в проекте *proj2* класс с именем *String2*, согласно текста листинга 2.8;
- 2) создать в проекте *proj2* класс с именем *BString*, согласно текста листинга 2.9;
- 3) провести разумную модификацию и тестирование работы созданных классов;
- 4) результаты исследования отразить в личном отчёте.

Листинг 2.8 — Исходный текст класса String2 проекта proj2

```
package rvs.pr3;
/**
 * Примеры использования класса String.
 * @author vgr
 */
public class String2 {

    public static void main(String[] args) {
        // №1 - Типичное создание строки
        String str1 = "    Первый пример создания строки    ";
        System.out.println("str1 = " + str1);

        // №2 - Создание строки на основе массива символов
        char[] char1 = {'П', 'е', 'р', 'в', 'ы', 'й'};
        String str2 = new String(char1);
        System.out.println("str2 = " + str2);

        // №3 - Создание пустого строкового объекта
        String str3 = new String();

        // №4 - Копирование строки str1 в новую строку str3
        str3 = str1;
        str1 = str1 + "добавка!";
        System.out.println("str1 = " + str1);
        System.out.println("str3 = " + str3);

        // №5 - Извлечение 5-го символа из строки str1
        char char2 = str1.charAt(5);
        System.out.println("char2 = " + char2);

        // №6 - Удаление всех пробелов в начале и конце строки
        str1 = str1.trim();

        // №7 - Выделение подстроки
        str3 = str1.substring(0,6);
        System.out.println("str3 = " + str3);

        // №8 - Сравнение строк
        if(str2.equals(str3))
            System.out.println("Строки str2 и str3 - равны");
        else
```

```

        System.out.println("Строки str2 и str3 - не равны");

        // №9 - Замена символа 'ы' на символ 'Ы' в строке str1
        str1 = str1.replace('ы', 'Ы');
        System.out.println("str1 = " + str1);

        // №10 - Разделение строки на слова и вычисление длины слов
        String[] str4 = str1.split(" ");
        for(int i=0; i<str4.length; i++)
            System.out.println(str4[i] + ": " + str4[i].length()
                               + " -> " + str4[i].getBytes().length);
    }
}

```

При исследовании приложения листинга 2.8 необходимо разобраться и описать все команды помеченные отдельными комментариями. Особое внимание уделить:

- 1) результату команд, помеченных комментарием №4;
- 2) подробно разобрать вывод команд, помеченных комментарием №10.

Листинг 2.9 — Исходный текст класса BString проекта proj2

```

package rvs.pr3;
/**
 * Примеры использования класса StringBuffer.
 * @author vgr
 */
public class BString {

    public static void main(String[] args) {
        // Пустой буфер
        StringBuffer sb1 = new StringBuffer();
        System.out.println("sb1 = " + sb1 + ", размер = " + sb1.length()
                           + ", буфер = " + sb1.capacity());

        // Пустой буфер размера 10 символов
        StringBuffer sb2 = new StringBuffer(10);
        System.out.println("sb2 = " + sb2 + ", размер = " + sb2.length()
                           + ", буфер = " + sb2.capacity());

        // Буфер инициализирован строкой
        StringBuffer sb3 =
            new StringBuffer(" Первый пример создания строки ");
        sb2 = sb3;
        System.out.println("sb2 = " + sb2 + ", размер = " + sb2.length());
        System.out.println("sb3 = " + sb3 + ", размер = " + sb3.length()
                           + ", буфер = " + sb3.capacity());

        // Добавление в начало строки
        sb3.insert(0, "<Начало строки>");
        System.out.println("sb2 = " + sb2 + ", размер = " + sb2.length());
        System.out.println("sb3 = " + sb3 + ", размер = " + sb3.length()
                           + ", буфер = " + sb3.capacity());

        // Добавление в конец строки
        sb3.append("<Конец строки>");
        System.out.println("sb2 = " + sb2 + ", размер = " + sb2.length());
        System.out.println("sb3 = " + sb3 + ", размер = " + sb3.length());
    }
}

```

```

        + ", буфер = " + sb3.capacity());

    // Реверс строки
    sb3.reverse();
    System.out.println("sb2 = " + sb2 + ", размер = " + sb2.length());
    System.out.println("sb3 = " + sb3 + ", размер = " + sb3.length()
        + ", буфер = " + sb3.capacity());
    }
}

```

При исследовании приложения листинга 2.9 необходимо разобраться и описать все команды класса **BString**. Особое внимание следует уделить вопросу: «Чем отличаются объекты *str1* и *str3* класса **String2** от соответствующих объектов *sb2* и *sb3* класса **BString**»?

На этом выполнение лабораторной работы №4 — заканчивается. Студент должен был получить все необходимые практические навыки для выполнения последующих работ.

2.3 Лабораторная работа №5.

Ввод/вывод языка Java

Настоящая лабораторная работа посвящена изучению синтаксических конструкций пакета *java.io* языка Java, которые обеспечивают другие классы и их объекты широким набором средств потокового ввода/вывода информации различного назначения.

Учебная цель данной работы — изучение технологии создания и использования в приложениях языка Java программных средств пакета *java.io* в пределах теоретического материала изложенного в подразделе 2.4 учебного пособия [1]:

- 1) стандартный ввод/вывод;
- 2) классы потоков ввода;
- 3) классы потоков вывода.

Практическая цель работы — лабораторное закрепление учебного материала посредством реализации и практического исследования набора учебных примеров, демонстрирующих применение инструментальных программных средств пакета *java.io* языка Java.

Общее учебное задание — реализовать и исследовать в проекте *proj3* интегрированной среды разработки Eclipse IDE набор учебных примеров, обеспечивающих достижение учебной и практической целей лабораторной работы №5. Учебные задания данной работы выполнить и изложить в личном отчёте, в виде последовательности следующих пунктов:

- 1) классы и методы стандартного ввода/вывода;
- 2) байтовые потоки ввода/вывода средствами классов *InputStream* и *OutputStream*;
- 3) инструментальные средства класса *File*;
- 4) сериализация объектов и объектный ввод/вывод инструментальными средствами классов *ObjectInputStream* и *ObjectOutputStream*;
- 5) символьный ввод/вывод средствами классов *Reader* и *Writer*.

2.3.1 Классы и методы стандартного ввода/вывода

Учебный материал пункта 2.4.1 пособия [1] демонстрирует нам семантическую схему стандартного ввода/вывода рисунком 2.27, показывающим его доступность для пользовательских программ написанных на языках C/C++ и Java.

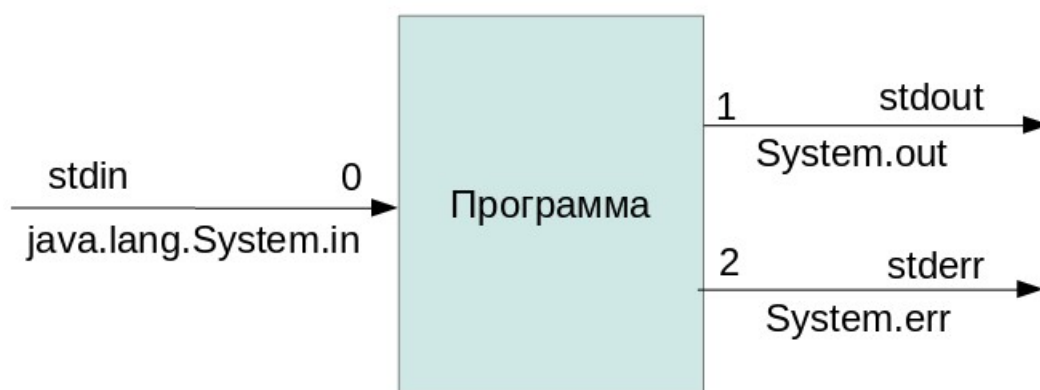


Рисунок 2.27 — Семантическая схема стандартного ввода/вывода

Что касается *стандартного вывода*, то мы его достаточно хорошо уже усвоили в предыдущих лабораторных работах и ещё уточним в следующем пункте данной работы.

Учебная цель данного пункта работы — практическое освоение программных технологий чтения байтовых потоков данных с клавиатуры ЭВМ средствами стандартного ввода языка Java.

Примечание — Следует помнить, что стандартный ввод данных с клавиатуры ЭВМ осуществляется через буфер устройства терминала операционной системы (ОС).

Байтовый потоковый стандартный ввод данных в языке Java осуществляется объектом *java.lang.System.in* типа *InputStream*. Семантическая схема взаимодействия этого объекта с устройством клавиатуры ЭВМ показана на рисунке 2.28.

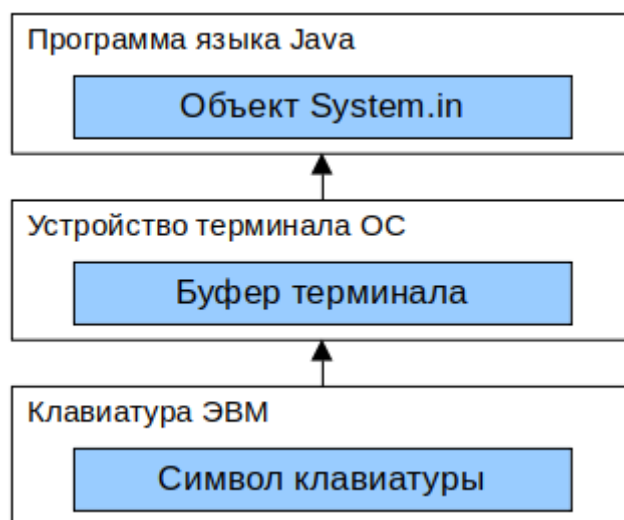


Рисунок 2.28 — Семантическая схема взаимодействия объекта *System.in* и клавиатуры ЭВМ

Семантическая схема представленного рисунка — достаточно проста:

- 1) каждый «нажатый» на клавиатуре символ попадает в буфер терминала и сохраняется там в виде байт, в соответствии с выбранной национальной кодировкой;
- 2) при нажатии на клавиатуре клавиши *<Enter>* содержимое буфера терминала передается в «активную» программу языка Java и полученная последовательность байт становится доступной для чтения её объектом *System.in*.

Объектам типа *InputStream* доступно множество различных методов чтения байтовых потоков, обеспеченных средствами языка Java. Наиболее важные из них приведены ниже в таблице 2.4.

Первое учебное задание:

- 1) создать проект с именем *proj3*;
- 2) создать в проекте *proj3* класс с именем *Example4*, согласно текста листинга 2.10;
- 3) создать в проекте *proj3* класс с именем *Example5*, согласно текста листинга 2.11;
- 4) разобрать алгоритмы работы созданных классов и провести их тестирование;
- 5) результаты исследования отразить в личном отчёте.

Таблица 2.4 — Наиболее важные методы класса `InputStream`

Метод	Описание метода
<code>public int read() throws IOException</code>	Читает по одному байту из потока ввода и возвращает целое число в пределах значений от 0 до 255.
<code>public int read(byte[] b) throws IOException</code>	Читает содержимое потока ввода и помещает его в массив байт <i>b</i> . Возвращает число прочитанных байт.
<code>public int read(byte[] b, int off, int len) throws IOException</code>	Читает <i>len</i> байт из потока ввода и помещает его в массив байт <i>b</i> , со смещением <i>off</i> . Возвращает число прочитанных байт.
<code>available()</code>	Возвращает число байт, доступных для чтения в потоке ввода.
<code>skip(long N)</code>	Пропускает во входном потоке <i>N</i> байт.
<code>close()</code>	Закрывает входной поток ввода.

Листинг 2.10 — Исходный текст класса *Example4* проекта *proj3*

```
package ru.tusur.asu;

import java.io.IOException; // Для обработки исключения

public class Example4 {

    public static void main(String[] args) {
        // Определение целочисленной рабочей переменной
        int myKey;

        System.out.println("Для выхода из программы, нажмите:\n" +
            "Ctrl-Z в DOS/Windows\n" +
            "Ctrl-D в UNIX/Linux");

        try
        {
            while ((myKey = System.in.read()) != -1)
            {
                System.out.println((char)myKey + ": " + myKey);
            }
        } catch(IOException e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

Первый пример, реализованный в виде класса *Example4*, использует самый простой метод `read()`, который:

- 1) читает по одному байту из входного потока и возвращает целочисленное значение этого байта в виде целого числа типа *int*;
- 2) если входной поток данных — пуст, то метод ожидает появление хотя бы одного байта данных;
- 3) если входной поток данных закрывается (при нажатии комбинации клавиш Ctrl-D), то метод `read()` возвращает значение *-1*.

Обратите также внимание на:

- 1) использование управляющего оператора *while*, реализующего цикл обработки;

- 2) использование обработчика исключений вида *try/catch*.

Второй пример, реализованный в виде класса *Example5*, использует более продвинутый способ чтения данных, динамически выделяя для хранения полученных данных байтовый массив:

- 1) для определения количества байт, находящихся в потоке стандартного ввода, используется метод *available()*;
- 2) если данных во входном потоке нет, то их ожидание осуществляется циклом организованным управляющим оператором *while*; это приводит к непроизводительному использованию ресурсов процессора, но обеспечивает целостную обработку строки байт переданных программ из буфера терминала.

Листинг 2.11 — Исходный текст класса *Example5* проекта *proj3*

```
package ru.tusur.asu;

import java.io.IOException; // Для обработки исключения

public class Example5 {

    public static void main(String[] args) {

        byte[] b; // Объявление байтового массива
        int n;    // Переменная хранения размера массива

        System.out.println("Программа Example5 делает 5 циклов:\n");
        try
        {
            for (int i = 0; i < 5; i++)
            {
                // Ожидаем ввод и определяем число символов
                // в буфере терминала
                while ((n = System.in.available()) == 0) ;

                b = new byte[n]; // Создаем массив буфера b
                System.in.read(b); // Читаем символы в буфер b

                // Печатаем весь буфер
                System.out.println(new String(b));
            }
        }
        catch(IOException e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

Примечание — Приведённые два примера являются учебными и не претендуют на образцы программирования средствами языка Java.

Теперь перейдём к более общим вопросам организации ввода/вывода на языке Java.

2.3.2 Байтовые потоки ввода/вывода средствами классов `InputStream` и `OutputStream`

Согласно общим концепциям языка Java, классы построенные только на супер-классе *java.lang.Object* не имеют собственных средств организации ввода/вывода. Они должны использовать объекты классов *InputStream* и *OutputStream*, определённых в пакете *java.io*, или их потомков, чтобы иметь указанные средства.

Классическим применением операций потокового ввода/вывода в приложениях являются задачи связанные с обработкой файлов: копирование, перемещение и изменение содержимого файлов различных типов.

Учебная цель данного пункта работы — практическое освоение программных технологий пакета *java.io* связанных с низкоуровневой обработкой файлов методами классов *InputStream* и *OutputStream*.

Второе учебное задание:

- 1) создать в проекте *proj3* класс с именем *Example6*, согласно текста листинга 2.12;
- 2) создать в проекте *proj3* класс с именем *Example7*, согласно текста листинга 2.13;
- 3) разобраться с алгоритмами работы созданных классов и провести их тестирование;
- 4) результаты исследования отразить в личном отчёте.

Первый пример учебного задания демонстрирует приложение, читающее содержимое уже имеющегося файла *~/src/rvs/Example1.java*, используя объект потокового ввода типа *InputStream*. Поскольку выбранный файл имеет текстовый тип, то его содержимое выводится в окно терминала Eclipse IDE.

Листинг 2.12 — Исходный текст класса Example6 проекта proj3

```
package ru.tusur.asu;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

public class Example6 {

    public static void main(String[] args) {

        // Объявление буфера и счетчика читаемых байт
        byte[] b;
        int n;

        System.out.println("Демонстрация потока FileInputStream.\n"
            + "Чтение файла: /home/upk/src/rvs/Example1.java\n"
            + "-----\n");

        try{
            InputStream in =
                new FileInputStream("/home/upk/src/rvs/Example1.java");

            while ((n = in.available()) > 0) // Читаем в цикле
            {
                b = new byte[n];
                in.read(b);
            }
        }
    }
}
```

```

        System.out.print(new String(b));
    }

    in.close();        // Закрываем поток ввода
} catch (IOException e) // Обрабатываем исключение
{
    System.out.println(e.getMessage());
}
}
}

```

Обратите внимание на следующие особенности реализации класса *Example6*:

- 1) входной поток *InputStream* открывается конструктором класса *FileInputStream*;
- 2) класс *FileInputStream* может генерировать исключение *FileNotFoundException*, но это исключение не обрабатывается в программе, поскольку перекрывается исключением *IOException*, генерируемым классом *InputStream*;
- 3) чтение файла и вывод содержимого на терминал осуществляется в цикле, потому что большие файлы могут читаться только порциями, определяемыми ПО самой ОС;
- 4) размер читаемой порции определяется методом *available()* объекта типа *InputStream*.

Второй пример учебного задания демонстрирует приложение, которое сначала записывает информацию в файл с именем *~/src/Demo7.txt*, используя объект потокового вывода типа *OutputStream*, а затем читает содержимое этого файла, выводя его на экран терминала. В выбранный файл записываются только текстовые строки исключительно из учебных целей, позволяющих обеспечить вывод содержимого этого файла на терминал.

Для реализации этого примера использованы методы класса *OutputStream*, которые представлены в таблице 2.5.

Таблица 2.5 — Наиболее важные методы класса *OutputStream*

Метод	Описание метода
<i>public void write(int b)</i> <i>throws IOException</i>	Записывает по одному байту в поток вывода числа от 0 до 255.
<i>public void read(byte[] b)</i> <i>throws IOException</i>	Записывает содержимое массива байт <i>b</i> в поток вывода.
<i>public void read(byte[] b,</i> <i>int off, int len)</i> <i>throws IOException</i>	Записывает <i>len</i> байт в потока вывода из массива байт <i>b</i> , со смещением <i>off</i> .
<i>flush()</i>	Принудительный сброс данных из промежуточного буфера в поток вывода.
<i>close()</i>	Закрывает выходной поток вывода.

Листинг 2.13 — Исходный текст класса *Example7* проекта *proj3*

```

package ru.tusur.asu;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class Example7 {

```

```

public static void main(String[] args) {

    // Объявление буфера и счетчика читаемых байт
    byte[] b;
    int n;

    System.out.println("Демонстрация потока FileOutputStream.\n"
        + "Создание и чтение файла: /home/upk/src/demo7.txt \n"
        + "-----\n");

    try{

        OutputStream out =
            new FileOutputStream("/home/upk/src/demo7.txt");

        // Пишем строки в файл
        out.write("Привет всем друзьям!\n".getBytes());
        out.write("Мы написали программу записи в файл.\n".getBytes());
        out.write("Посмотрим, что из этого получится...\n".getBytes());
        out.flush();      // Сбрасываем поток вывода
        out.close();      // Закрываем поток вывода

        InputStream in =
            new FileInputStream("/home/upk/src/demo7.txt");

        while ((n = in.available()) > 0) // Читаем в цикле
        {
            b = new byte[n];
            in.read(b);
            System.out.print(new String(b));
        }

        in.close(); // Закрываем поток ввода

    }catch(IOException e)
    {
        System.out.println(e.getMessage());
    }
}

```

Обратите внимание на следующие особенности реализации класса *Example7*:

- 1) выходной поток *OutputStream* открывается конструктором класса *FileOutputStream*;
- 2) класс *FileOutputStream* может генерировать исключение *FileNotFoundException*;
- 3) текстовая строка представляет объект типа *String*, поэтому к ней сразу применяется метод *getBytes()*, обеспечивающий нужный тип данных для потока вывода;
- 4) перед закрытием выходной поток должен быть «сброшен» методом *flush()*, чтобы исключить потерю данных при записи в файл.

2.3.3 Инструментальные средства класса File

Излишне утверждать, что программисту любого языка программирования постоянно приходится работать как с файлами, так и с файловой системой в целом. Для этих целей пакет *java.io* имеет класс типа *File*, краткому изучению которого и посвящен данный пункт ла-

бораторной работы. Следует также заметить, что в языке Java с версии 1.4 появился ещё новый пакет *java.nio*, который поддерживает работу с различными буферами ввода/вывода, каналами и другими расширенными средствами обработки информации. Для более глубокого изучения технологий ввода/вывода языка Java студенту рекомендуется обращаться к более продвинутому источнику [3]:

- 1) пакету *java.io* посвящена глава 20;
- 2) пакету *java.nio* посвящена глава 21.

Третье учебное задание:

- 1) создать в проекте *proj3* класс с именем *Example8*, согласно текста листинга 2.14;
- 2) создать в проекте *proj3* класс с именем *File2*, согласно текста листинга 2.15;
- 3) провести тестирование созданных приложений (см. пояснения к выполнению примеров);
- 4) результаты исследования отразить в личном отчёте.

Первый пример учебного задания демонстрирует приложение, определяющее различные свойства файла *~/src/demo7.txt*, используя различные методы класса типа *File*, которые выводятся в окно терминала Eclipse IDE.

Этот пример должен показать широкие возможности используемого класса.

Листинг 2.14 — Исходный текст класса Example8 проекта proj3

```
package ru.tusur.asu;

import java.io.File;
import java.util.Date;

public class Example8 {

    public static void main(String[] args)
    {
        // Вычисляем и печатаем число аргументов программы
        int n = args.length;
        System.out.println("Число аргументов: " + n);

        String fname = "/home/upk/src/demo7.txt";
        if ( n > 0) fname = args[0];
        System.out.println("Используем имя файла: " + fname + "\n");

        // Определяем и создаём объект класса File
        File myf = new File(fname);

        System.out.println("Это - файл: " + myf.isFile());
        System.out.println("Это - директория: " + myf.isDirectory());

        System.out.println("Можно писать в файл: " + myf.canWrite());
        System.out.println("Можно читать файл: " + myf.canRead());
        System.out.println("Можно запускать файл: " + myf.canExecute());

        System.out.println("Имеет родителя: " + myf.getParent());
        System.out.println("Имеет путь: " + myf.getPath());
        System.out.println("Имеет имя: " + myf.getName());

        System.out.println("Длина файла: " + myf.length());
        System.out.println("Последняя модификация: "
```

```

        + new Date(myf.lastModified()));
    }
}

```

Обратите внимание, что текст класса *Example8* демонстрирует определение числа аргументов запускаемой программы, и если такие аргументы — есть, то имя исследуемого файла заменяется на значение первого аргумента.

Чтобы освоить полные возможности класса *Example8*, следует:

- 1) создать в каталоге *~/lib* архивный файл проекта *proj3* с именем *proj3.8.jar*, указав в качестве запускаемого приложения имя класса *Example8*;
- 2) запустить окно терминала, перейти в каталог *~/lib* и запускать приложение командой представленной выражением (2.15).

java -jar Example8 Строка_аргумента (2.15)

Примечание — Изучаемый класс типа *File* содержит достаточно богатый набор конструкторов и методов, обеспечивающих программное управление файлами и каталогами.

В одной лабораторной работе отсутствует возможность изучения полных возможностей класса *File*, поэтому ограничимся только следующими моментами:

- 1) изучаемый класс содержит четыре конструктора, синтаксис и семантика которых представлена в таблице 2.6;
- 2) пакет *java.io* автоматически определяет используемую ОС и предоставляет программисту статические константы для разделения путей и имён файлов, синтаксис и семантика которых представлена в таблице 2.7;
- 3) отдельные возможности методов изучаемого класса показаны в примере класса *File2*, а остальной потенциал класса следует изучать по источнику [3].

Таблица 2.6 — Конструкторы класса *File* пакета *java.io* языка Java

Конструктор	Описание конструктора
<i>File (File parent, String child)</i>	Создаёт новый экземпляр типа <i>File</i> из родительского абстрактного пути и строки дочернего пути.
<i>File (String pathname)</i>	Создаёт новый экземпляр файла путём преобразования заданной строки пути в абстрактный путь.
<i>File (String parent, String child)</i>	Создаёт новый экземпляр <i>File</i> из строки родительского пути и строки дочернего пути.
<i>File (URI uri)</i>	Создаёт новый экземпляр <i>File</i> путём преобразования указанного <i>URI</i> -файла в абстрактный путь.

Таблица 2.7 — Поля-разделители пакета *java.io* языка Java

Конструктор	Описание конструктора
<i>static String pathSeparator</i>	Системно зависимый символ-разделитель пути, представленный для удобства в виде строки.
<i>static char pathSeparatorChar</i>	Системно зависимый символ-разделитель пути.
<i>static String separator</i>	Системно зависимый символ разделителя имён по умолчанию, представленный для удобства в виде строки.
<i>static char separatorChar</i>	Системно зависимый символ разделителя имён по умолчанию.

Второй пример учебного задания демонстрирует интерактивное приложение реализованное в виде класса *File2*, которые взаимодействует с пользователем в окне собственного терминала среды разработки Eclipse IDE.

Студенту следует самостоятельно определить новые используемые методы и прокомментировать их применение в личном отчёте.

Листинг 2.15 — Исходный текст класса File2 проекта proj3

```
package rvs.pr1;

import java.io.File;
import java.io.IOException;
import java.io.PrintStream;

/**
 * Дополнительный пример по использованию класса File.
 * @author vgr
 */
public class File2
{
    public static void main(String[] args) throws IOException
    {
        PrintStream msg =
            System.out;
        // Получаем домашнюю директорию пользователя
        String home =
            System.getenv("HOME");
        msg.println("Домашняя директория: " + home);

        msg.println("Разделитель пути: " + File.pathSeparator);
        msg.println("Разделитель имен: " + File.separator);

        File files = new File(home);
        msg.println("URI: " + files.toURI().toString());
        msg.print("Для продолжения - нажми Enter ...");
        System.in.read();

        // Получаем список всех файлов.
        String[] fs = files.list();

        msg.println("Список каталогов:");
        for(int i=0; i<fs.length; i++)
        {
            File ff = new File(home, fs[i]);
            if(ff.isDirectory())
                msg.println("\t" + fs[i]);
        }
        msg.print("Для продолжения - нажми Enter ...");
        System.in.read();

        msg.println("Список файлов:");
        for(int i=0; i<fs.length; i++)
        {
            File ff = new File(home, fs[i]);
            if(ff.isFile())
                msg.println("\t" + fs[i]);
        }
    }
}
```



```

    }
    msg.print("Для продолжения - нажми Enter ...");
    System.in.read();

    // Создаем каталог $HOME/src2
    File dir = new File(home, "src2");

    msg.print("Директория: " + dir.getPath());
    if(dir.mkdir())
        msg.println(" - создана!");
    else
        msg.println(" - не создана!");

    // Удаляем созданный каталог
    msg.print("Проверь наличие каталога и нажми Enter ...");
    System.in.read();

    if(dir.delete())
        msg.println("\n\nКаталог - удален!");
    else
        msg.println("\n\nКаталог - не удален!");
}
}

```

2.3.4 Сериализация объектов и объектный ввод/вывод инструментальными средствами классов `ObjectInputStream` и `ObjectOutputStream`

Язык Java является языком ООП. Базовая технология программной работы с объектами этого языка предполагает:

- 1) создание объекта *с помощью конструктора* нужного типа;
- 2) целевые изменения объекта с помощью имеющихся у объекта *методов*;
- 3) *уничтожение объекта* («сборка мусора») по мере необходимости или при завершении работы приложения, в котором этот объект использовался.

Многие прикладные задачи требуют, чтобы создаваемые приложениями объекты сохранялись и в дальнейшем могли использоваться другими приложениями. Такими задачами являются:

- 1) *асинхронное по времени взаимодействие приложений*, обменивающиеся данными в виде объектов;
- 2) *сетевые приложения*, использующие обмен данных в виде объектов;
- 3) *объектные распределённые системы*, являющиеся основным предметом изучения в данной дисциплине.

Программный пакет ***java.io*** предоставляет программистам все необходимые инструменты для решения указанных выше задач:

- 1) средствами интерфейса ***java.io.Serializable***;
- 2) средствами классов типа ***java.io.ObjectInputStream*** и ***java.io.ObjectOutputStream***.

Учебная цель данного пункта работы — практическое освоение программных технологий пакета ***java.io*** обеспечивающих сериализацию создаваемых классов языка Java, сохранение объектов этих классов в файловой системе ОС и последующее использование сохранённых объектов.

Четвёртое учебное задание:

- 1) создать в проекте *proj3* класс с именем *Mesg1*, согласно текста листинга 2.16, реализующий интерфейс *Serializable* и предназначенный для сохранения его объектов в среде файловой системы ОС;
- 2) создать в проекте *proj3* класс с именем *Serial1*, согласно текста листинга 2.17, реализующий сохранение в файле *~/src/Serial1.dat* объектов типа *Mesg1* с помощью потока объектного вывода *ObjectOutputStream*;
- 3) создать в проекте *proj3* класс с именем *Serial2*, согласно текста листинга 2.18, реализующий чтение из файла *~/src/Serial1.dat* объектов типа *Mesg1* с помощью потока объектного ввода *ObjectInputStream*;
- 4) разобраться с технологией функционирования этих классов, а также провести тестирование их работы;
- 5) результаты исследования отразить в личном отчёте.

Первый пример учебного задания демонстрирует приложение в виде *сериализуемого класса* типа *Mesg1*, объекты которого предназначены для сохранения в файле или других целях, требующих его представления специальной последовательностью байт и возможностью последующего использования в других приложениях.

Содержательная часть каждого объекта типа *Mesg1* сохраняется в целочисленной переменной *n* и строковой переменной *msg*. Методы *getN()* и *getMsg()* обеспечивают извлечение указанных данных из экземпляра класса.

Листинг 2.16 — Исходный текст класса *Mesg1* проекта *proj3*

```
package rvs.pr2;

import java.io.Serializable;
import java.util.Date;

public class Mesg1 implements Serializable
{
    /**
     * Стандартная константа версии класса
     */
    private static final long serialVersionUID = 1L;

    // Сохраняемые данные класса
    int n = 0;
    String msg = "";

    // Конструктор класса
    Mesg1 (int i, String text)
    {
        this.n = i;
        this.msg = "Сообщение №" + String.valueOf(i) + ": "
        + new Date().toString()
        + "\n" + text
        + "-----";
    }

    // Методы чтения сохраняемых данных класса
    public int getN()
    {
        return this.n;
    }
}
```

```

    }

    public String getMsg()
    {
        return this.msg;
    }
}

```

Обратите внимание, что:

- 1) сериализуемые классы могут интегрировать в себе объекты любой сложности, а не только простые объекты типа *int* и *String*;
- 2) все интегрированные в класс объекты также сериализуются;
- 3) не все типы классов могут быть подвергнуты сериализации, поэтому если их включить в целевой класс, то его сериализация становится невозможной.

Второй пример учебного задания демонстрирует приложение, реализованное в виде класса *Serial1*:

- 1) создающее в интерактивном режиме объекты типа *Mesg1*, с помощью конструктора этого класса;
- 2) записывающее созданные объекты в файл *~/src/Serial1.dat*, с помощью объектного потокового вывода типа *ObjectOutputStream*.

Непосредственная реализация класса *Serial1* опирается на конструктор и методы, представленные в следующей таблице 2.8.

Таблица 2.8 — Описание конструктора и базовых методов класса *ObjectOutputStream*

Конструктор/метод	Описание конструктора/метода
<i>ObjectOutputStream (OutputStream поток_вывода) throws IOException</i>	Конструктор, создающий <i>объектный поток вывода</i> на основе уже существующего байтового потока вывода. В качестве потока вывода реально используется объект: <i>new FileOutputStream(имя_файла)</i>
<i>final void writeObject (Object объект)</i>	Записывает заданный объект в созданный объектный поток вывода.
<i>void flush ()</i>	Делает конечным состояние вывода, очищая все буфера, в том числе и буфера вывода.
<i>void close()</i>	Системно зависимый символ разделителя имён по умолчанию.
<i>Другие методы вывода</i>	Другие <i>методы вывода простых типов данных</i> , например, <i>void write(int b)</i> , <i>void writeFloat(float f)</i> и другие подобные методы.

Листинг 2.17 — Исходный текст класса *Serial1* проекта *proj3*

```

package rvs.pr2;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

/**
 * Класс, записывающий объекты типа Mesg1 в файл.
 * @author vgr
 */
public class Serial1
{

```

```

// Метод main()
public static void main(String[] args)
{
    // Имя файла записи объектов
    String file = System.getenv("HOME")
        + File.separator + "src"
        + File.separator + "Serial1.dat";

    // Объектная переменная записываемого объекта
    Mesg1 msg;

    int n = 0; // Номер сообщения
    int m = 0; // Число байт в сообщении
    byte[] buf; // Буфер сообщения

    // Создаём объектный поток вывода
    try (ObjectOutputStream oos =
        new ObjectOutputStream(new FileOutputStream(file)))
    {
        // Цикл ввода и записи данных и объекта
        System.out.println("Пустое сообщение - завершение программы!");
        while ( n > -1)
        {
            System.out.println("Введи новое сообщение:");
            while ( System.in.available() == 0 ) ;

            m = System.in.available();
            System.out.println("Длина сообщения: " + m);
            if ( m < 2 ) break;
            buf = new byte[m];
            System.in.read(buf);

            // Создание передаваемого объекта
            n++;
            msg = new Mesg1(n, new String(buf));

            // Запись объекта в поток вывода
            oos.writeObject(msg);
        }
        // Выталкиваем данные в поток и закрываем его
        oos.flush();
        oos.close();

        System.out.println("Проверь наличие и содержание файла: "
            + file);
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}
}

```

Обратите внимание, что:

- 1) интерактивный режим приложения *Serial1* осуществляется в окне терминала Eclipse IDE;
- 2) если на приглашение «Введи новое сообщение:» просто нажать клавишу «Enter», то работа программы завершится;

- 3) объект выходного потока *oos* определён как аргумент оператора *try/catch*, что позволяет автоматически закрыть его в случае возникновения исключения;
- 4) необходимо также проверить факт создания целевого файла.

Третий пример учебного задания демонстрирует приложение, реализованное в виде класса *Serial2*:

- 1) читающее объекты типа *Mesg1* из файла *~/src/Serial1.dat*, с помощью объектного потока ввода типа *ObjectInputStream*;
- 2) извлекающее из объектов сообщения с помощью метода *getMsg()* и выводящее их в окно терминала Eclipse IDE.

Непосредственная реализация класса *Serial2* опирается на конструктор и методы, представленные в следующей таблице 2.9.

Таблица 2.9 — Описание конструктора и базовых методов класса *ObjectInputStream*

Конструктор/метод	Описание конструктора/метода
<i>ObjectInputStream (InputStream поток_вывода) throws IOException</i>	Конструктор, создающий <i>объектный поток ввода</i> на основе уже существующего байтового потока ввода. В качестве потока ввода реально используется объект: <i>new FileInputStream(имя_файла)</i>
<i>final Object readObject ()</i>	Читает объект из входного потока.
<i>int available ()</i>	Возвращает количество байтов, доступных в данный момент в буфере ввода.
<i>void close()</i>	Системно зависимый символ разделителя имён по умолчанию.
<i>Другие методы ввода</i>	Другие <i>методы ввода простых типов данных</i> , например, <i>int read()</i> , <i>double readFloat()</i> и другие подобные методы.

Листинг 2.18 — Исходный текст класса *Serial2* проекта *proj3*

```
package rvs.pr2;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

/**
 * Класс, читающий объекты класса Mesg1 из файла.
 * @author vgr
 */
public class Serial2 {

    public static void main(String[] args)
    {
        // Имя файла чтения объектов
        String file = System.getenv("HOME")
            + File.separator + "src"
            + File.separator + "Serial1.dat";

        // Объектная переменная целевого объекта
        Mesg1 msg;
        int n = 0; // Число прочитанных сообщений

        // Создаем объектный поток ввода
        try (ObjectInputStream ois =
```

```

        new ObjectInputStream(new FileInputStream(file)))
    {
        // Цикл чтения объектов
        while ( n > -1)
        {
            // Читаем объект из файла
            msg =
                (Mesg1)ois.readObject();
            n++;

            // Печатаем сохраненное сообщение объекта.
            System.out.println(msg.getMsg());
        }
        ois.close();
    }
    catch (IOException e)
    {
        //e.printStackTrace();
        System.out.println("Прочитано " + n + " сообщений!");
    }
    catch (ClassNotFoundException e2)
    {
        e2.printStackTrace();
        System.out.println("e2");
    }
}
}

```

Обратите внимание, что:

- 1) метод **readObject()** возвращает объект типа **Object**, поэтому его результат в явном виде переводится в тип **Mesg1**;
- 2) обычно в одном файле сохраняется один объект типа, который потом и читается;
- 3) поток **ObjectInputStream** не имеет методов прогнозирующих наличие объектов в потоке ввода, поэтому приложение **Serial2** заканчивает своё функционирование по исключению **IOException**.

2.3.5 Символьный ввод/вывод средствами классов Reader, Writer

Байтовые потоки ввода/вывода, основанные на классах **InputStream** и **OutputStream** обеспечивают решение всех задач, связанных с программированием чтения и записи произвольных *сериализованных типов объектных данных*, но вызывают затруднения, когда необходимо обрабатывать символьные данные, предполагающие использование национальных языков и различных кодировок символов.

Для оптимизации работы с символьными данными пакет **java.io** предлагает абстрактные классы **java.io.Reader** и **java.io.Writer**. На их основе созданы дополнительные подклассы, ориентированные на специальные направления разработки приложений:

- 1) в таблице 2.10 представлен перечень подклассов абстрактного класса **Reader**;
- 2) в таблице 2.11 представлен перечень подклассов абстрактного класса **Writer**.

Учебная цель данного пункта работы — практическое освоение программного применения классов **InputStreamReader** и **OutputStreamWriter**, входящих в пакет **java.io** языка Java.

Таблица 2.10 — Перечень подклассов абстрактного класса `java.io.Reader`

Класс	Описание класса
<i>BufferedReader</i>	Определяет возможности использования буферизованных потоков ввода.
<i>LineNumberReader</i>	Определяет дополнительные возможности считывания из потоков ввода строк с учётом их нумерации.
<i>CharArrayReader</i>	Определяет возможности использования символьного потока на основе массивов данных.
<i>FilterReader</i>	Определяет набор действий обработки фильтруемых потоков ввода.
<i>PushBackReader</i>	Определяет возможность возвращения обратно в поток ввода считанного символа.
<i>InputStreamReader</i>	Определяет возможности трансляции (перевода) байтовых потоков ввода в символьные.
<i>FileReader</i>	Определяет возможности использования в качестве потоков ввода файлов на жёстком диске компьютера.
<i>PipedReader</i>	Определяет возможности использования канальных потоков ввода символов.
<i>StringReader</i>	Определяет возможности использования строк в потоках ввода.

Таблица 2.11 — Перечень подклассов абстрактного класса `java.io.Writer`

Класс	Описание класса
<i>BufferedWriter</i>	Класс определяет возможности использования буферизованных символьных потоков вывода.
<i>CharArrayWriter</i>	Класс определяет возможности использование потока для записи данных в символьный массив.
<i>FilterWriter</i>	Класс определяет набор действий обработки фильтруемых потоков вывода.
<i>OutputStreamWriter</i>	Определяет возможности перевода символьных потоков вывода в байтовые.
<i>FileWriter</i>	Определяет возможности вывода символьных потоков в файлы.
<i>PipedWriter</i>	Класс определяет возможности вывода символьных потоков в каналы.
<i>PrintWriter</i>	Определяет возможности использования методов <i>println(...)</i> и <i>print(...)</i> для вывода информации.
<i>StringWriter</i>	Класс определяет возможности записи символьных строк в потоки вывода.

Пятое учебное задание:

- 1) создать в проекте *proj3* класс с именем *ReadChars*, согласно текста листинга 2.19, демонстрирующий потоковое символьное чтение данных с клавиатуры посредством использования класса *InputStreamReader*;
- 2) создать в проекте *proj3* класс с именем *WriteChars*, согласно текста листинга 2.20, демонстрирующего различный результат использования метода *write(char c)* объекта типа *System.out* и объекта символьного потока класса *OutputStreamWriter*;
- 3) провести тестирование работы созданных приложений;
- 4) результаты работы отразить в личном отчёте.

Первый пример учебного задания демонстрирует приложение *ReadChars*, аналогичное по алгоритму реализации приложению *Example4*, но читающее по одному символу с клавиатуры методом *read()* из символьного потока *InputStreamReader* и выводящее этот символ на печать методом *print()* стандартного объекта *System.out*.

Листинг 2.19 — Исходный текст класса *ReadChars* проекта *proj3*

```
package rvs.pr3;

import java.io.IOException;
```

```

import java.io.InputStreamReader;

/**
 * Демонстрация работы потока InputStreamReader
 * @author vgr
 */
public class ReadChars
{
    public static void main(String[] args) {
        // Определение целочисленной рабочей переменной
        int mKey;

        System.out.println("Для выхода из программы, нажмите:\n" +
            "Ctrl-Z в DOS/Windows\n" +
            "Ctrl-D в UNIX/Linux");

        try (InputStreamReader isr =
            new InputStreamReader(System.in))
        {
            // Используемая кодировка
            System.out.println("Используемая кодировка: " +
                isr.getEncoding());

            // Цикл чтения и разбора символов
            while ((mKey = isr.read()) != -1)
            {
                if( mKey == 13 )
                {
                    System.out.println(mKey + " - Возврат каретки");
                    continue;
                }
                if( mKey == 10 )
                {
                    System.out.println(mKey + " - Перевод строки");
                    continue;
                }
                System.out.print(mKey + " - "
                    + (char)mKey + ", ");
            }
            isr.close(); // Закрываем поток
            System.out.println("Завершение работы программы...");
        }
        catch(IOException e)
        {
            System.out.println(e.getMessage());
        }
    }
}

```

Второй пример учебного задания демонстрирует приложение *WriteChars*, которое в цикле разбирает на символы текст четырех вариантов строк, вычисляет количество символов в них и число байт, а затем посимвольно выводит эти строки на экран терминала методом *write(char c)*, в двух вариантах: посредством объекта *System.out* и посредством объекта символического потока данных *OutputStreamWriter*.

Листинг 2.20 — Исходный текст класса *WriteChars* проекта *proj3*

```
package rvs.pr3;

import java.io.IOException;
import java.io.OutputStreamWriter;

/**
 * Демонстрация работы потока OutputStreamWriter
 * @author vgr
 */
public class WriteChars {

    public static void main(String[] args) {
        // Строки на английском/русском языках
        String[] str = {"Hello, world!\n",
            "Пример использования русских символов\n",
            "Console\n",
            "Совет!\n"};

        // Открытие потока вывода OutputStreamWriter
        try (OutputStreamWriter osw =
            new OutputStreamWriter(System.out))
        {
            // Цикл обработки
            for(int n=0; n < str.length; n++)
            {
                // Вывод System.out
                System.out.print("В строке " + str[n].length()
                    + " символов=" + str[n].getBytes().length
                    + " байт: " + str[n]);

                System.out.print("Метод out.write: ");
                for(int i=0; i < str[n].length(); i++)
                    System.out.write(str[n].charAt(i));

                System.out.print("Метод osw.write: ");
                System.out.flush();

                // Вывод OutputStreamWriter
                for(int i=0; i < str[n].length(); i++)
                    osw.write(str[n].charAt(i));
                osw.flush();
            }
            osw.close();
        }
        catch(IOException e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

Этими примерами завершается лабораторная работа №5. Считается, что выполнив эту работу, студент освоил основы разработки приложений, обеспечивающих ввод/вывод средствами пакета *java.io* языка Java.

2.4 Лабораторная работа №6. Сокеты и сетевое ПО языка Java

Настоящая лабораторная работа посвящена изучению синтаксических конструкций пакета *java.net* языка Java, которые обеспечивают разработку сетевых приложений различного назначения.

Учебная цель данной работы — изучение технологии создания и использования в приложениях языка Java программных средств пакета *java.net* в пределах теоретического материала изложенного в подразделе 2.5 учебного пособия [1]:

- 1) адресация на базе класса *InetAddress*;
- 2) адресация на базе *URL* и *URLConnection*;
- 3) сокеты протокола TCP;
- 4) сокеты протокола UDP;
- 5) простейшая задача технологии клиент-сервер.

Практическая цель работы — лабораторное закрепление учебного материала посредством реализации и практического исследования набора учебных примеров, демонстрирующих применение инструментальных программных средств пакета *java.net* языка Java.

Общее учебное задание — реализовать и исследовать в проектах интегрированной среды разработки Eclipse IDE набор учебных примеров, обеспечивающих достижение учебной и практической целей лабораторной работы №6. Учебные задания данной работы выполнить и изложить в личном отчёте, в виде последовательности следующих пунктов:

- 1) реализация сетевой адресации объектами классов *InetAddress*, *URL* и *URLConnection*;
- 2) реализация сетевого приложения в виде класса *TCPServer*;
- 3) реализация сетевого приложения в виде класса *TCPClient*.

2.4.1 Реализация сетевой адресации объектами классов *InetAddress*, *URL* и *URLConnection*

Многие современные приложения, включая распределённые вычислительные системы (РВС), работают в сети Интернет, взаимодействуя друг с другом. Сам факт такого взаимодействия предполагает наличие в любой сети ЭВМ специальной адресации, обеспечивающей целевое взаимодействие. Инструментальный пакет *java.net* языка Java содержит классы, обеспечивающие сетевое взаимодействие на базе стека протоколов TCP/IP и соответствующую адресацию абонентов сети на основе протоколов IPv4, IPv6 и HTTP.

В целом студентам хорошо известна адресация перечисленных выше протоколов, поскольку она широко используется в сети Интернет и изучается в соответствующих дисциплинах, но язык Java, как язык ООП, использует адреса как объекты классов. Это требует как теоретическое, так и практическое освоение классов адресации сетевых приложений.

Учебная цель данного пункта работы — практическое освоение программных технологий, использующих классы *InetAddress*, *URL* и *URLConnection* языка Java.

Первое учебное задание:

- 1) создать в среде Eclipse IDE проект с именем *proj4*;

- 2) создать в проекте *proj4* класс с именем *Example9*, согласно текста листинга 2.21, демонстрирующее использование объектов класса *InetAddress*;
- 3) создать в проекте *proj4* класс с именем *Example10*, согласно текста листинга 2.22, демонстрирующее использование объектов классов *URL* и *URLConnection*;
- 4) провести исследование и тестирование реализованных приложений;
- 5) результаты исследования отразить в личном отчёте.

Первый пример учебного задания основан на учебном материале пункта 2.5.1 учебного пособия [1]. Он использует *три статических метода* класса *InetAddress*, создающие три смысловых типа объектов сетевых адресов:

- 1) *getLocalHost()* — создаёт сетевой адрес на основе настроек ОС ЭВМ;
- 2) *getByName(String host)* — создаёт объект адреса по строковому адресу хоста;
- 3) *getAllByName(String host)* — создаёт массив объектов адреса по строковому адресу хоста.

Примечание — Строковое имя *host* может быть как цифровым, так и доменным. Если имя — доменное, то производится обращение к серверу имён и, если доменное имя не найдено, то возникает исключение *UnknownHostException*.

Другие два не статических метода класса *InetAddress* возвращают:

- 1) *getHostAddress()* — строковое значение цифрового адреса объекта класса *InetAddress*;
- 2) *getHostName()* — строковое доменное имя объекта класса *InetAddress*, если доменное имя отсутствует, то возвращается строка цифрового имени.

Листинг 2.21 — Исходный текст класса *Example9* проекта *proj4*

```
package ru.tusur.asu;

import java.net.InetAddress;
import java.net.UnknownHostException;

public class Example9 {

    public static void main(String[] args) {

        System.out.println("Использование класса InetAddress.\n"
                           + "-----");

        try {
            // Создаем объекты адресов
            InetAddress addr1 =
                InetAddress.getLocalHost();
            InetAddress addr2 =
                InetAddress.getByName("asu.tusur.ru");
            InetAddress[] addr3 =
                InetAddress.getAllByName("www.yandex.ru");

            // Выводим на печать содержимое объектов
            System.out.println("Локальная ЭВМ: "
                               + addr1.getHostAddress() + " | " + addr1.getHostName());

            System.out.println("Кафедра АСУ : "
                               + addr2.getHostAddress() + " | " + addr2.getHostName());
```

```

        for(int i=0; i < addr3.length; i++)
            System.out.println("Яндекс      : "
                               + addr3[i].getHostAddress() + " | "
                               + addr3[i].getHostName());

        System.out.println("Нормальное завершение...");
    }
    catch(UnknownHostException e)
    {
        System.out.println(e.getMessage());
        System.out.println("\nАварийное завершение...");
    }
}
}

```

Второй пример учебного задания основан на учебном материале пункта 2.5.1 учебного пособия [1]. Создаваемый в приложении *Example10* объект адреса типа *URL* имеет метод *openStream()*, применение которого создаёт объект потокового байтового типа *InputStream*. Использование этого потока позволяет читать ресурсы WWW серверов.

Примечание — Перед запуском приложения следует подмонтировать и запустить сервер приложений Apache TomEE.

Листинг 2.22 — Исходный текст класса Example10 проекта proj4

```

package ru.tusur.asu;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;

public class Example10 {

    public static void main(String[] args) {

        // Объявление буфера и счетчика читаемых байт
        byte[] b;
        int n;

        // Целевой адрес для URL
        //String address = "http://asu.tusur.ru/";
        String address = "http://localhost:8080/";           // Адрес Apache TomEE

        System.out.println("Использование класса URL\n"
                           + "Сохраняем в файле: /home/upk/src/demo10.html\n"
                           + "-----");

        try
        {
            // Объявление и создание объекта типа URL
            URL url =
                new URL(address);

```

```

// Печатаем компоненты URL-адреса
System.out.println("Протокол: " + url.getProtocol() + "\n"
    + "Хост      : " + url.getHost() + "\n"
    + "Порт      : " + url.getPort() + "\n"
    + "Файл       : " + url.getFile() + "\n"
    + "Полная форма: " + url.toExternalForm() + "\n");
InputStream in = // Открываем входной поток
    url.openStream();
OutputStream out = // Открываем выходной поток
    new FileOutputStream("/home/upk/src/demo10.html");

// Исключительно для демонстрации
URLConnection ucon = url.openConnection();
ucon.connect();

while ((n = in.available()) > 0) // Читаем и пишем в цикле
{
    b = new byte[n];
    in.read(b); // Читаем
    out.write(b); // Пишем
    // Дублируем на экран терминала
    System.out.print(new String(b));
}

in.close(); // Закрываем потоки
out.close();
System.out.println("Приложение завершило работу...");
}
catch(MalformedURLException e1)
{
    System.out.println(e1.getMessage());
}
catch(IOException e2)
{
    System.out.println(e2.getMessage());
}
}
}

```

2.4.2 Реализация сетевого приложения в виде класса TCPServer

В основе архитектуры всех сетевых приложений лежит базовая обобщённая модель «клиент- сервер»:

- 1) **клиент** — основная программа, которая предназначена для достижения конкретной прикладной цели;
- 2) **сервер** — вспомогательная программа, которая предназначена для обслуживания потребностей одной или нескольких программ-клиентов.

Учебная цель данного пункта работы — практическое освоение технологии реализации программ-серверов средствами пакета *java.net* языка Java.

Второе учебное задание:

- 1) создать в среде Eclipse IDE проект с именем *proj5*;
- 2) создать в проекте *proj5* класс с именем *TCPServer*, согласно текста листинга 2.23, демонстрирующего технологию реализации простейшей программы-сервера;
- 3) выполнить тестирование запуска программы-сервера в командной строке терминала;

- 4) выполнить тестирование запуска программы-сервера инструментальными средствами проекта *proj5* в среде разработки Eclipse IDE;
- 5) результаты исследования отразить в личном отчёте.

Алгоритм работы реализуемого приложения соответствует учебному материалу пункта 2.2.5 пособия [1]. Для удобства понимания и реализации класса *TCPServer* его базовая семантика вынесена в аннотацию исходного текста программы.

Листинг 2.23 — Исходный текст класса *TCPServer* проекта *proj5*

```
package asu.server;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Date;

/*
 * Сервер стартует и прослушивает порт.
 * Сервер акцептирует запрос на соединение, создает потоки ввода/вывода
 * и в цикле выполняет:
 * 1) читает входной поток посимвольно и выводит их на экран;
 * 2) получив символ "\n" == 10, посылает клиенту строку "OK\n";
 * 3) получив символ "\r" == 13, посылает его назад клиенту;
 * 4) завершает работу по любому исключению или закрытию входного потока.
 */

public class TCPServer {

    public static void main(String[] args)
    {
        // Объявляем объектные и другие рабочие переменные
        int port; // Номер порта сервера
        String smes= "OK\n"; // Сообщение посылаемое клиенту

        ServerSocket svrsocket; // Серверный сокет
        Socket clientsocket; // Клиентский сокет

        // Время получения первого пакета
        long statime = new Date().getTime();

        // Текущее время относительно акцепта соединения
        long curtime = 0;

        InputStream in; // Входной поток сервера
        OutputStream out; // Выходной поток сервера

        // Печать цифровых значений управляющих символов
        int ch = (int)'\n'; // Символ конца строки
        System.out.println("\n\n = " + ch);
        int cr = (int)'\r'; // Символ синхронизации
        System.out.println("\r = " + cr + "\n");

        try{
            // Читаем аргументы запускаемой программы
```

```

port = // Читаем номер порта как аргумент программы
        new Integer(args[0]).intValue();

// Определяем и печатаем параметры запуска сервера
InetAddress localhost = // Адрес сервера
        InetAddress.getLocalHost();
System.out.println("Server Address: "
        + localhost.getHostAddress());
System.out.println("Port Address: "
        + port); //args[0]);

// Запускаем сервер и печатаем время старта
svrsocket = new ServerSocket(port);
System.out.println("TCPserver start: " + new Date());

// Запускаем прослушивание заданного порта
clientsocket = svrsocket.accept(); // Ожидаем соединения
statime = new Date().getTime();
System.out.print(curtime +
        ": TCP-server - accept conection...\n\n" + "0: ");

// Открываем байтовые потоки ввода/вывода
in = clientsocket.getInputStream();
out = clientsocket.getOutputStream();

// Цикл диалога с клиентом
// Читаем по байту, пока не закроется входной поток сокета
while((ch=in.read()) != -1){

    System.out.print((char)ch);
    // Перевод строки - значит строка закончилась
    if(ch == 10){
        curtime = new Date().getTime();
        System.out.print((curtime - statime) + ": ");

        // Отвечаем клиенту "OK\n"
        out.write(smes.getBytes());
    }
    // Возврат каретки - значит синхронизация с клиентом
    if(ch == 13) out.write("\r".getBytes());

}

out.flush(); // Освобождаем поток вывода
out.close(); // Закрываем потоки
in.close(); //
clientsocket.close(); // Закрываем сокеты
svrsocket.close(); //

}
catch(UnknownHostException ue)
{
    System.out.println("UnknownHostException: " + ue.getMessage());
}
catch(IOException e)
{
    System.out.println("IOException: " + e.getMessage());
}
catch(Exception ee)
{

```

```

        System.out.println("Exception: " + ee.getMessage());

        // Подсказка для запуска программы
        System.out.println("\nrun: java asu.server.TCPServer port\n");
    }

    curtime = new Date().getTime(); // Подводим итог
    System.out.println("\n" + (curtime - statime)
        + ": TCP-server - stop");

    System.exit(0); // Системное завершение программы
}
}

```

Обратите внимание, что если сервер запускается без единственного аргумента — номер порта, то сервер не запускается и выводится следующее сообщение: `"\nrun: java asu.server.TCPServer port\n"`.

Студенту следует провести тестирование запуска сервера в двух вариантах: *из командной строки терминала* и *из среды Eclipse IDE*.

Из среды терминала сервер запускается командой (2.16), как это показано на рисунке 2.29.

`java asu.server.TCPServer 8888` (2.16)

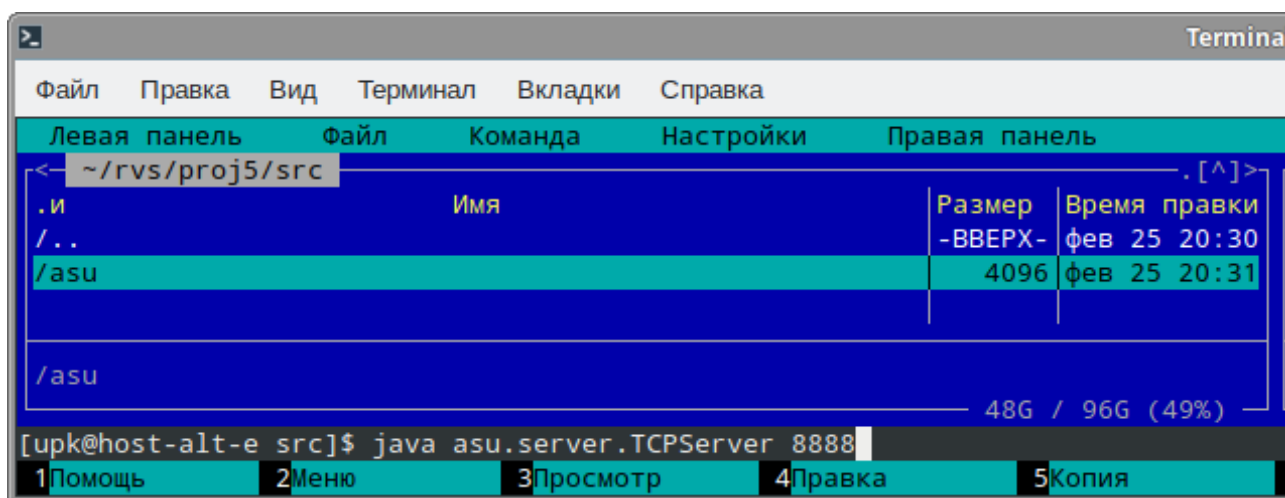


Рисунок 2.29 — Запуск сервера из командной строки терминала

Из среды **Eclipse IDE** сервер запускается обычным способом, но предварительно нужно настроить среду запуска для этого приложения «*Run Configurations*».

Чтобы выполнить нужную настройку «*Run Configurations*», необходимо:

- 1) выделить в «*Package Explorer*» среды Eclipse IDE файл настраиваемого приложения, как это показано на рисунке 2.30;
- 2) активировать контекстное меню (правой кнопкой мышки) и выбрать «*Run As/Run Configurations ...*»;
- 3) запустится нужное окно «*Run Configurations*», в котором следует сделать нужные настройки, как это показано на рисунке 2.31

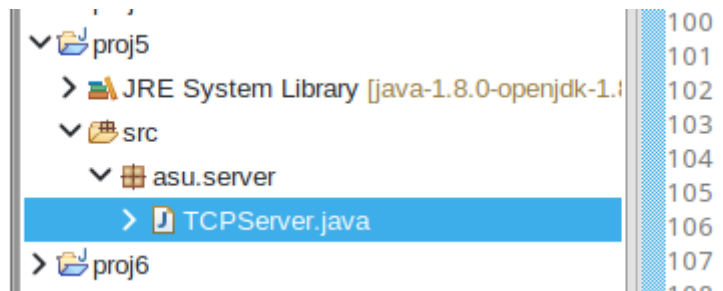


Рисунок 2.30 — Выделение приложения, требующего настройки в «Run Configurations»

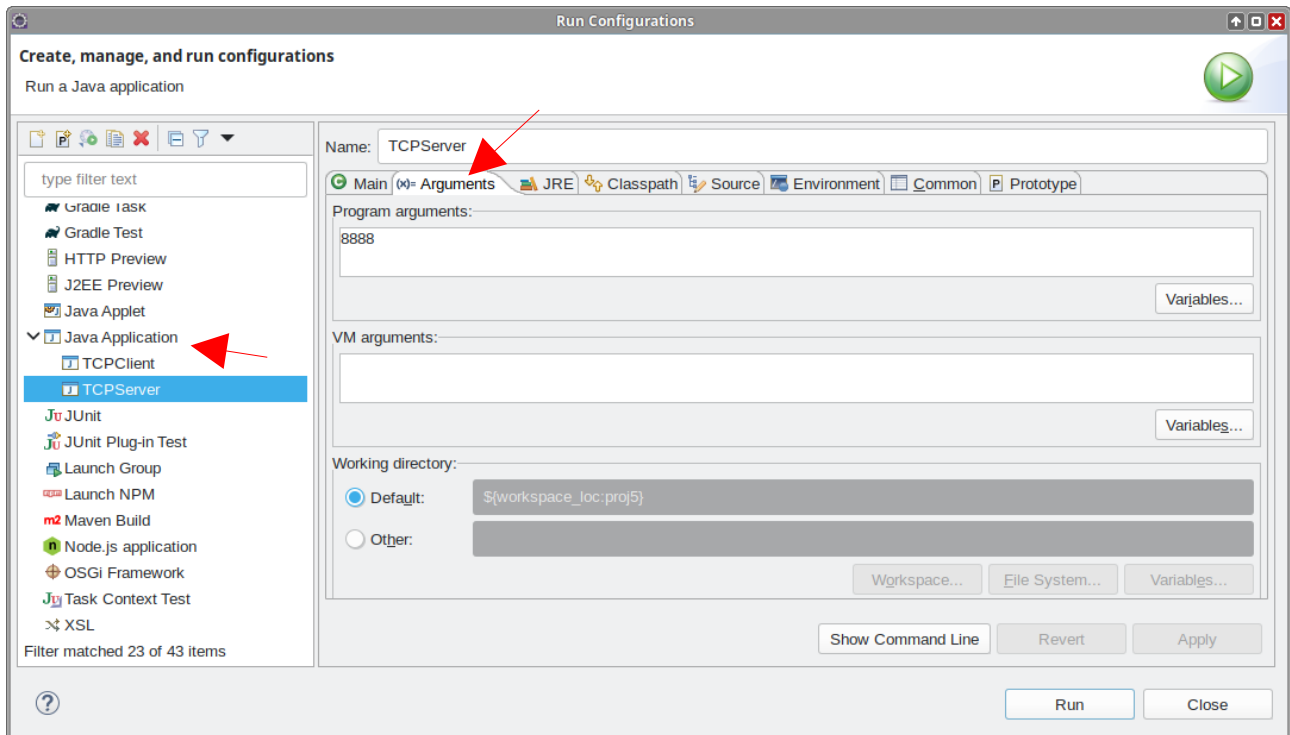


Рисунок 2.31 — Задание аргументов на запуск приложению TCPServer

Примечание — Настраиваемое приложение ранее не запускалось на выполнение, то оно может не отражаться в окне «Run Configurations».

Если окно «Run Configurations» не отражает имя нужного класса, то необходимо сделать следующее (см. рисунок 2.31):

- 1) дважды «кликнуть» мышкой, в левой части окна по пункту «Java Application», тогда должны появиться все имена классов, к которым применимо понятие настройки;
- 2) выделяем мышкой нужное имя класса (**TCPServer**) и переходим к правой части окна;
- 3) поле «Name:» должно содержать нужное нам имя класса;
- 4) выделяем вкладку «(x)= Arguments»;
- 5) в поле «Program arguments» вводим нужные аргументы *столбиком*;
- 6) если аргумент состоит из нескольких слов, то заключаем его в *двойные кавычки*;
- 7) не забудьте сохранить настройки кнопкой «Apply»;
- 8) приложение можно сразу запустить кнопкой «Run».

2.4.3 Реализация сетевого приложения в виде класса TCPClient

Типичная ситуация, возникающая при реализации сетевого приложения согласно модели «клиент-сервер», состоит в том, что:

- 1) **программа-сервер** обычно обеспечивает *множество программ-клиентов* некоторым набором данных и её хорошая работа определяется скоростью и надёжностью поставленной достаточно узкой задачи;
- 2) **программа-клиент** обычно обеспечивает функционалом (обслуживает) *конкретного конечного пользователя*; именно по качеству этого обслуживания и даётся оценка качества работы всего сетевого приложения.

Из сказанного выше следует, что в паре сетевого взаимодействия именно программа-клиент является *активной стороной* обеспечивающей:

- 1) контроль наличия соединения с сервером;
- 2) контроль синхронизации процессов обмена данными с сервером.

Примечание — Алгоритмическая проблема реализации программы-клиента — адекватный *контроль процессов чтения данных с сервера*.

Проблема контроля чтения данных программой-клиентом вызвана *следующими причинами*, обусловленными особенностями реализации инструментального ПО ЭВМ:

- 1) *возможный разрыв соединения* с сервером не обязательно отражается на состоянии буфера чтения (потока ввода) программы-клиента;
- 2) *пустой буфер чтения* программы-клиента не отражает истинного состояния соответствующей программы-сервера, которая сама может находиться в состоянии чтения;
- 3) *чтение из пустого буфера* программы-клиента блокирует (останавливает) выполнение этой программы;
- 4) *запись данных в буфер вывода* программы-клиента однозначно сигнализирует о разрыве соединения с сервером, но не гарантирует правильную обработку пересылаемых данных программой сервером.

Перечисленные выше причины заявленной проблемы определяют и общую схему алгоритма контроля чтения данных программой-клиентом:

- 1) **определение тайм-аута**, который адекватен времени приемлемого периода ожидания приёма данных от сервера;
- 2) **определение сообщения синхронизации**, которое посылается серверу при истечении времени определённого тайм-аута;
- 3) **ожидание ответа** на сообщение синхронизации;
- 4) **принятие решения** о продолжении взаимодействия с сервером или завершение работы программой-клиентом.

Инструментальные средства языка Java предлагают нам два подхода реализации указанной выше схемы контроля чтения данных:

- 1) многопоточное программирование посредством использования интерфейса **Runnable** и методы класса **Thread**;
- 2) метод **available()** входного потока типа **InputStream**.

В данной работе мы не будем использовать технологии многопоточного программирования, поскольку их изучение выходит за рамки нашей дисциплины. Студенты, желающие самостоятельно изучить эти технологии, могут прочитать главу 11 источника [3].

Что касается метода `available()` входного потока типа *InputStream*, то его применение уже известно студенту по результатам предыдущей лабораторной работы. Эта причина и является основанием для применения его в приложении данного пункта работы.

Учебная цель данного пункта работы — практическое освоение технологии реализации *программ-клиентов* средствами пакета *java.net* языка Java.

Третье учебное задание:

- 1) создать в среде Eclipse IDE проект с именем *proj6*;
- 2) создать в проекте *proj6* класс с именем *TCPClient*, согласно текста листинга 2.24, демонстрирующего технологию реализации простейшей программы-клиента;
- 3) выполнить тестирование запуска программы-клиента в командной строке терминала;
- 4) выполнить тестирование запуска программы-клиента инструментальными средствами проекта *proj6* в среде разработки Eclipse IDE;
- 5) результаты исследования отразить в личном отчёте.

Примечание — Программа-клиент должна запускаться после успешного запуска программы-сервера.

Алгоритм работы реализуемого приложения соответствует учебному материалу пункта 2.2.5 пособия [1]. Для удобства понимания и реализации класса *TCPClient* его базовая семантика вынесена в аннотацию исходного текста программы.

Листинг 2.24 — Исходный текст класса TCPClient проекта proj6

```
package asu.client;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Date;

/*
 * Клиент читает параметры командной строки:
 * адрес сервера, порт сервера, количество посылаемых сообщений, текст сообщения.
 * Клиент устанавливает соединение, создает потоки ввода/вывода
 * и в цикле выполняет:
 * 1) выводит на экран то, что посылает: "<номер сообщения>" + "<сообщение>";
 * 2) выводит на экран символ входной поток символов;
 * 3) получив символ "\n", переходит к п.1;
 * 4) если входной поток пуст более dTime, то серверу посылается символ '\r';
 * 5) получив от сервера символ '\r', то посылает пакет повторно;
 * 6) завершает работу по любому исключению; после передачи заданного
 * количества сообщений или когда входной канал - пуст; тайм-аут
 * ожидания подтверждения больше dTime после reset (посылки серверу
 * символа '\r').
 */
```

```

public class TCPCClient {

    public static void main(String[] args)
    {
        // Объявляем объектные и другие рабочие переменные
        InetAddress remotehost; // Адрес сервера
        int port = 0;           // Номер порта сервера
        int nn = 0;             // Количество сообщений
        String mes = " ";       // Сообщение клиента

        Socket clientsocket;    // Объявление клиентского сокета

        long curtime = 0;
        long dTime = 100;
        long rTime = 0;
        boolean flagRepeat = false; // Нормальное ожидание ответа сервера

        InputStream in;
        OutputStream out;

        // Печать цифровых значений управляющих символов
        int ch = (int)'\n';
        System.out.println("\n = " + ch);
        int cr = (int)'\r';
        System.out.println("\r = " + cr + "\n");

        try{
            System.out.println("\nTCP-client - start: " + new Date());

            // Чтение и инициализация аргументов программы
            remotehost = InetAddress.getByName(args[0]);
            System.out.println("Server Address: "
                               + remotehost.getHostAddress()); //args[0]);

            port = new Integer(args[1]).intValue();
            System.out.println("ServerPort Address: "
                               + port); //args[1]);

            dTime = new Integer(args[2]).longValue();
            System.out.println(" Time_out (msec): "
                               + args[2]);

            nn = new Integer(args[3]).intValue();
            System.out.println("Count Message: "
                               + args[3]);

            mes = args[4];
            System.out.println("Client Message: "
                               + args[4] + "\n");

            // Устанавливаем соединение с сервером
            clientsocket=new Socket(remotehost, port);
            System.out.println("TCPclient connect: " + new Date());
            long statime = new Date().getTime();

            // Создание потоков ввода/вывода
            in = clientsocket.getInputStream();
            out = clientsocket.getOutputStream();
        }
    }
}

```

```

// Цикл диалога с сервером
for(int i=1; i<=nn; i++){
    // Посылаем серверу сообщение
    mes = String.valueOf(i) + " " + args[4] + "\n";
    out.write(mes.getBytes());

    // Фиксируем границу тайм-аута
    rTime = new Date().getTime() + dTime;
    curtime = new Date().getTime();
    System.out.println((curtime - statime) + ": " + mes);

    // Ожидаем ответ сервера
    boolean flag = true;           // Когда - цикл ожидания
    while(flag)
    {
        if(in.available() > 0)    // Есть символы в буфере ввода
        {
            flagRepeat = false;   // Ожидания синхронизации нет

            // Блокирующая операция чтения одного байта
            ch = in.read();
            System.out.print((char)ch);
            if(ch == 10)           // Если пришло подтверждение
            {
                curtime = new Date().getTime();
                System.out.print((curtime - statime) + ": ");
                flag = false;
                flagRepeat = false;
            }
            if(ch == 13){          // Если пришла синхронизация
                // Заново отправляем пакет
                out.write(mes.getBytes());
                rTime = new Date().getTime() + dTime;
            }
        }
        else                      // Нет символов в буфере ввода
        {
            System.out.print(".");

            // Проверка тайм-аута
            if(new Date().getTime() > rTime)
            {
                // Принятие решения, когда тайм-аут - исчерпан
                if(flagRepeat) // Завершаем работу программы
                {
                    flag = false;
                }
                else           //Инициация синхронизации
                {
                    out.write("\r".getBytes());
                    rTime = new Date().getTime() + dTime;
                    flagRepeat = true;
                }
                // Конец принятия решения
            }
        } // Конец оператора if/else
    } // Конец цикла while

    if(flagRepeat) break;        // Выходим из цикла for
}

```

```

    } // Конец цикла for
    // Сбрасываем и закрываем потоки
    out.flush();
    out.close();
    in.close();
    clientsocket.close(); // Закрываем сокет

} catch (UnknownHostException ue)
{
    System.out.println("\nUnknownHostException: "
        + ue.getMessage());

} catch (IOException e)
{
    System.out.println("\nIOException: " + e.getMessage());

} catch (Exception ee)
{
    System.out.println("\nException: " + ee.getMessage());

    // Подсказка для запуска программы
    System.out.println("\nrun: java asu.client.TCPClients address "
        + "port time_out count_message message\n");
}

System.out.println("\nTCPclient stop: " + new Date());
System.exit(0);
}
}

```

Обратите внимание, что если *программа-клиента* запускается без аргументов или имеются другие ошибки задания аргументов, то **TCPClient** не запускается и выводится следующее сообщение: `"\nrun: java asu.client.TCPClients address port time_out count_message message\n"`.

Студенту следует провести тестирование запуска программы-клиента в двух вариантах: *из командной строки терминала* и *из среды Eclipse IDE*.

Из среды терминала программа-клиент запускается из каталога `~/rvs/proj6/bin`, например, командой (2.17).

```

java asu.client.TCPClient localhost 8888 1000 100 \
    "Text message from client program"

```

(2.17)

Примечание — Задание аргументов программы клиента должно определяться результатов вывода программы-сервера.

Из среды Eclipse IDE программа-клиент запускается обычным способом, но предварительно нужно настроить среду запуска для этого приложения «*Run Configurations*» также, как это было сделано в предыдущем пункте для программы-сервера.

Этими примерами завершается лабораторная работа №6. Считается, что выполнив эту работу, студент освоил разработку простых сетевых приложений средствами пакета **java.net** языка Java.

2.5 Лабораторная работа №7.

Технология работы с СУБД Apache Derby

Настоящая лабораторная работа является завершающей в серии из пяти работ по второй теме изучаемой дисциплины «*Инструментальные средства языка Java*». Она посвящена изучению синтаксических конструкций пакета *java.sql* языка Java, которые обеспечивают разработку приложений широкого класса информационных систем. С прикладной точки зрения данная лабораторная работа завершает базовое обучение студентов языку Java, которое является вполне достаточным для успешного изучения и практического освоения последующих тем настоящей дисциплины.

Учебная цель данной работы — изучение технологии создания и использования в приложениях языка Java программных средств пакета *java.sql* в пределах теоретического материала изложенного в подразделе 2.6 учебного пособия [1] — «Организация доступа к базам данных»:

- 1) инструментальные средства СУБД Apache Derby;
- 2) SQL-запросы и драйверы баз данных;
- 3) типовой пример выборки данных.

Практическая цель работы — лабораторное закрепление учебного материала посредством реализации набора учебных примеров, демонстрирующих применение инструментальных программных средств пакета *java.sql* языка Java.

Общее учебное задание — реализовать и исследовать в проектах интегрированной среды разработки Eclipse IDE набор учебных примеров, обеспечивающих достижение учебной и практической целей лабораторной работы №7. Учебные задания данной работы выполнить и изложить в личном отчёте, в виде последовательности следующих пунктов:

- 1) изучение инфраструктуры размещения СУБД Apache Derby;
- 2) создание учебной базы данных *exampleDB*;
- 3) реализация учебного примера *Example11*.

Примечание — При выполнении данной лабораторной работы студенту необходимо строго следовать требованиям и рекомендациям изложенным в данном подразделе учебного пособия.

Столь строгое требование обосновано рядом изменений используемых в лабораторной работе инструментальных языка Java. Это значительно сократит временные затраты студента, которые необходимы для её успешного выполнения.

2.5.1 Изучение инфраструктуры размещения СУБД Apache Derby

Любая СУБД является основным инструментом разработчика информационных систем, поэтому необходимо хорошо инфраструктуру размещения этого инструмента.

Учебная цель данного пункта работы — изучение инфраструктуры размещения компонент СУБД Apache Derby, которая полностью реализована на языке Java.

Первое учебное задание:

- 1) перечитать содержимое пункта «1.2.2 Тестирование ПО СУБД Apache Derby» данного учебного пособия, чтобы вспомнить контекст выполняемой работы;

- 2) проверить настройки переменных среды в файле `~/.bashrc`, которые должны соответствовать информации, представленной ранее [на рисунке 1.11](#);
- 3) запустить на исполнение сценарий `~/bin/mountDerby`, который монтирует дистрибутив Apache Derby к рабочему каталогу `/opt/derby`;
- 4) проверить работоспособность ПО Apache Derby;
- 5) сделать доступной для изучения официальную документацию СУБД Apache Derby;
- 6) результаты исследования отразить в личном отчёте.

Поскольку первые три пункта не требуют пояснений, обсудим выполнение четвертого и пятого.

Проверка работоспособности ПО Apache Derby.

Запустим в терминале файловый менеджер и перейдём в каталог `/opt/derby`, как это показано на рисунке 2.32. Хорошо видно, что дистрибутив Apache Derby успешно монтирован в свой домашний каталог.

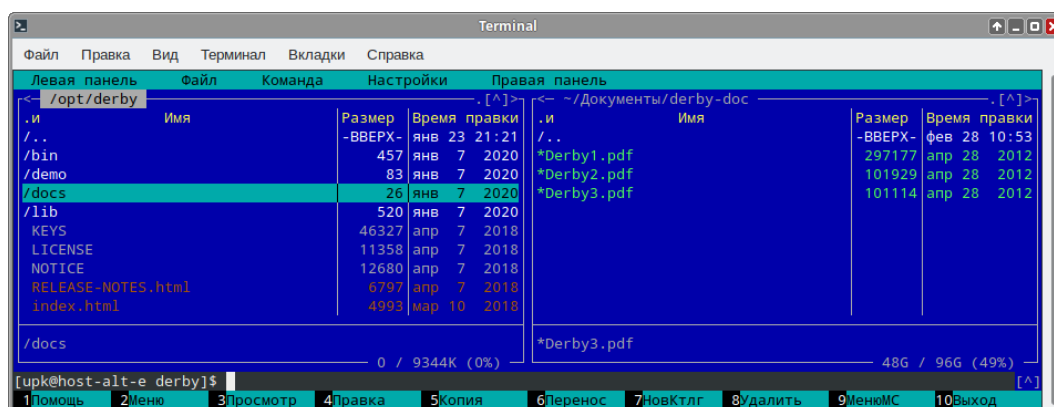


Рисунок 2.32 — Каталоги монтирования СУБД Derby и документации на неё

Теперь в командной строке терминала выполним команду выражения (2.18). Результат должен быть подобным тому, что показано на рисунке 2.33.

java org.apache.derby.tools.sysinfo (2.18)

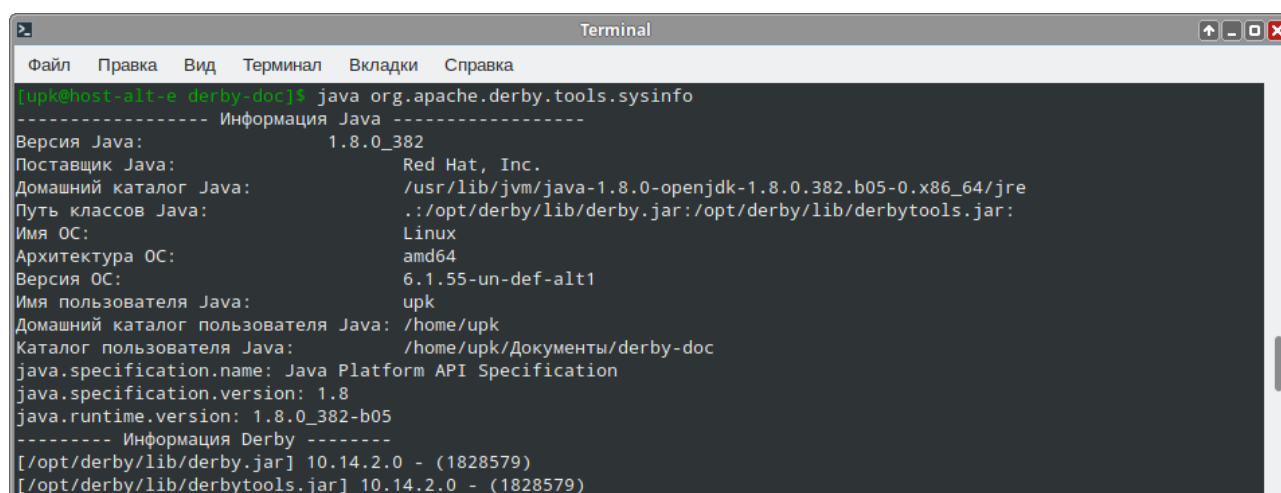


Рисунок 2.33 — Успешный результат выполнения команды выражения (2.18)

Примечание — Команда выражения (2.18) является официальным тестом работоспособности ПО Apache Derby.

Использование официальной документации СУБД Apache Derby.

Дистрибутив Apache Derby, как и дистрибутив любой СУБД, имеет полный набор документации, правда на английском языке. Естественно, сто студент должен владеть навыками работы с ней. Как показано ранее [на рисунке 2.32](#), в правом окне файлового менеджера каталог `~/Документация/derby-doc` содержит три файла в формате **pdf**, где на русском языке изложено мнение сотрудников корпорации IBM по следующим вопросам:

- 1) **Derby1.pdf** — «Проект Apache Derby»: общее описание проекта, история его возникновения и другие вопросы;
- 2) **Derby2.pdf** — «Введение в системы реляционных баз данных Derby»: многие вопросы инсталляции дистрибутива, настройки среды ОС и другие аспекты;
- 3) **Derby3.pdf** — «Введение в JDBC Derby».

В целом указанные статьи интересны не только для общего ознакомления, но и содержат много познавательной информации.

На том же [рисунке 2.32](#) (в левом окне файлового менеджера) показано размещение официальной документации дистрибутива. Это — каталог **docs** и два файла: **index.html** и **RELEASE-NOTES.html**.

Примечание — Каталог `/opt/derby` монтирован только на чтение, поэтому нормальное чтение документации — проблематично.

Следует скопировать указанный каталог и файлы в другое место, например, в каталог `~/Документация/derby-doc`, как показано на рисунке 2.34.

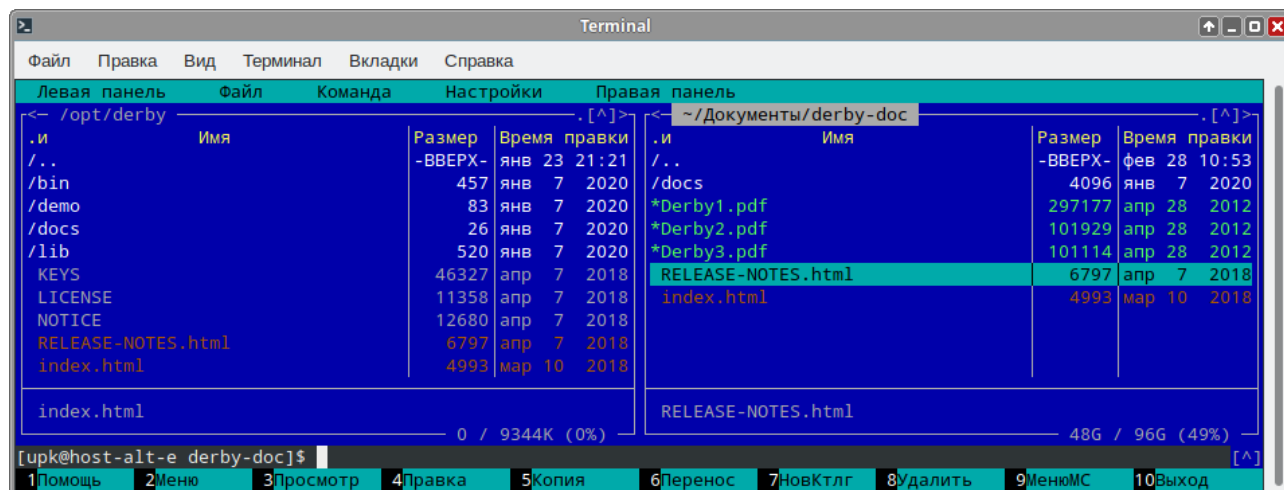


Рисунок 2.34 — Размещение официальной документации в рабочей области студента

Примечание — Из официальной документации дистрибутива удалена документация в формате **html**, но оставлена документация в формате **pdf**. Тем не менее любую документацию лучше размещать на личном flashUSB с целью экономии рабочего пространства пользователя **upk**.

Если мышкой активировать файл **RELEASE-NOTES.html**, то запустится браузер, как это показано на рисунке 2.35.

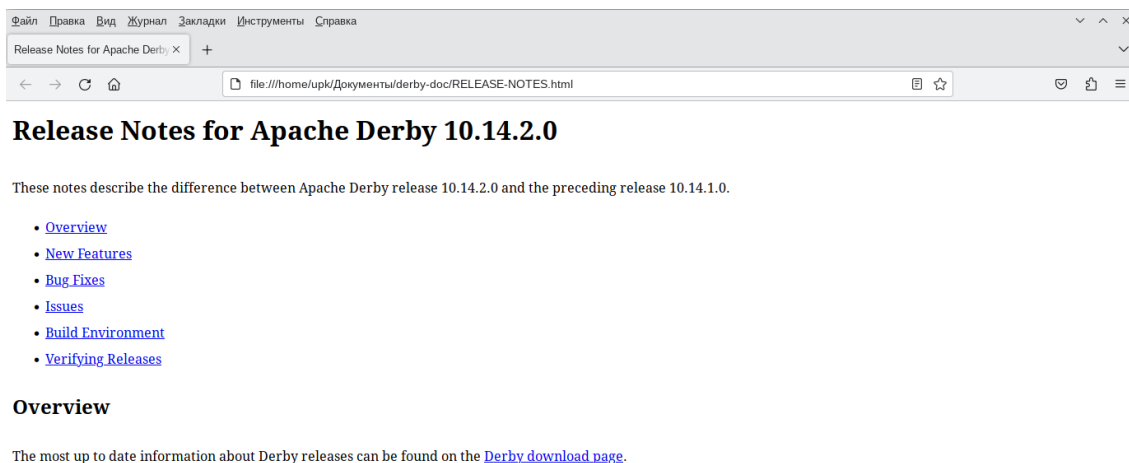


Рисунок 2.35 — Отображение файла RELEASE-NOTES.html в окне браузера

Если мышкой активировать файл *index.html*, то браузер запустит страницу доступа к документации Apache Derby, как это показано на рисунке 2.36.

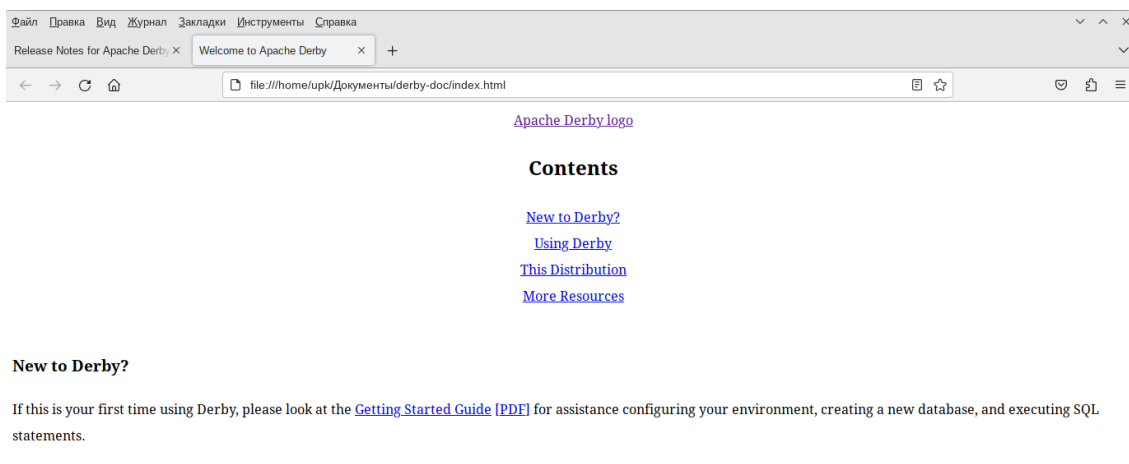


Рисунок 2.36 — Страница доступа к локальной документации дистрибутива Apache Derby

Если на странице рисунка 2.35 перейти по ссылке «*[PDF]*», в окне браузера запустится соответствующая документация в формате *pdf*, как показано на рисунке 2.37.

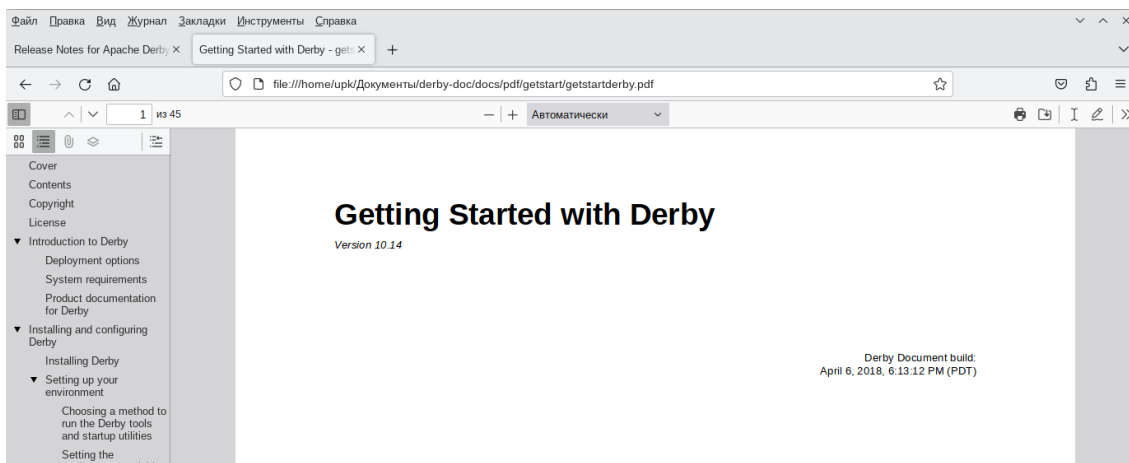


Рисунок 2.37 — Доступ к документации Apache Derby в формате pdf

Примечание — Каталог `~/Документация/derby-doc/docs` (рисунок 2.34) содержит документацию в формате *pdf*, поэтому её можно запускать прямо из окна файлового менеджера.

Студенту следует исследовать каталог *docs* и познакомиться с содержимым документации Apache Derby. Результат исследования отразить в личном отчёте.

2.5.2 Создание учебной базы данных *exampleDB*

Как данная лабораторная работа, так и учебный материал последующих разделов дисциплин требует использования баз данных и работу с их таблицами. С целью сокращения общего объема вспомогательных работ и лучшего сосредоточения на главной тематике изучаемой дисциплины принято решение об использовании единой базы данных:

- 1) база данных имеет имя *exampleDB*;
- 2) имя используемой таблицы *notepad*;
- 3) метод доступа к базе данных — *Embedded*: встроенный вариант доступа с использованием драйвера «*org.apache.derby.jdbc.EmbeddedDriver*».

Примечание — Хотя учебный материал данной работы опирается на содержание подраздела 2.6 учебного пособия [1], следует строго придерживаться требований именно данных методических указаний.

Учебная цель данного пункта работы — получение навыков создания баз данных инструментальными средствами СУБД Apache Derby и интегрированной среды разработки Eclipse IDE.

Второе учебное задание:

- 1) создать базу данных *exampleDB* и таблицу *notepad*, используя сценарий языка SQL содержащийся в файле `~/databases/createDB.sql`;
- 2) проверить правильность создания базы данных сценарием `~/databases/selectDB.sql`;
- 3) создать в среде Eclipse IDE проект с именем *proj7*;
- 4) создать в проекте *proj7* класс с именем *CreateDB*, согласно текста листинга 2.25, демонстрирующего технологию создания баз данных средствами пакета *java.sql* на языке Java;
- 5) проверить правильность создания базы данных сценарием `~/databases/selectDB.sql`;
- 6) результаты исследования отразить в личном отчёте.

Выполнение представленного задания рекомендуется выполнять в заданной последовательности пунктов. Это значительно экономит время на его успешное исполнение.

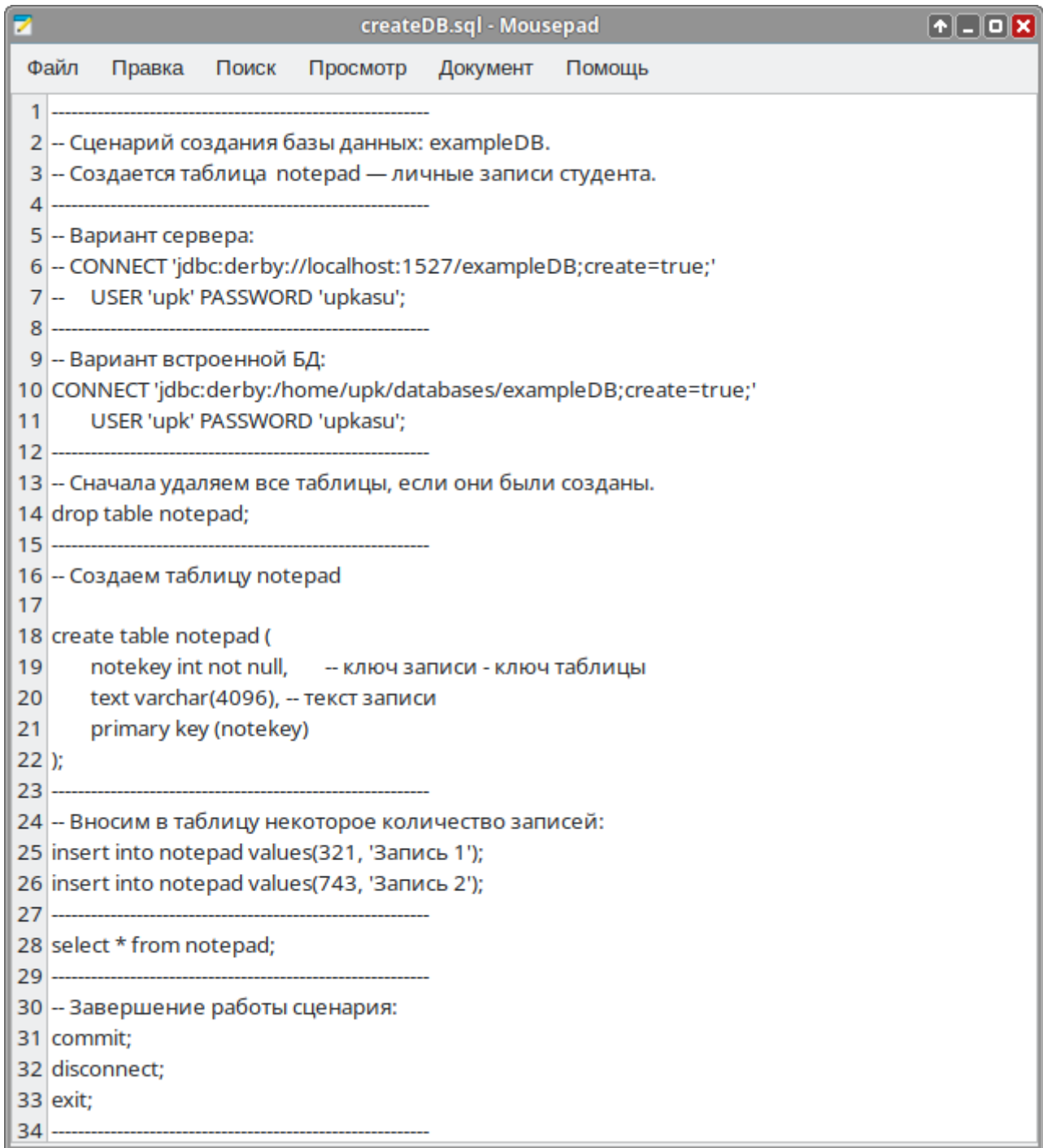
Создание базы данных средствами сценария *createDB.sql*.

Сначала следует запустить виртуальный терминал с файловым менеджером, перейти в каталог `~/databases` и убедиться, что в нем отсутствует директория *examoleDB*. Если такая директория присутствует, то её следует удалить вместе с содержимым.

Затем нужно проверить содержимое файла *createDB.sql*, текст которого должен совпадать с текстом рисунка 2.38. Особое внимание следует уделить:

- 1) оператору *CONNECT*, который определяет: вариант доступа к базе данных, местоположение создаваемой базы данных, имя и пароль, с которыми будет осуществляться доступ к базе данных;

- 2) оператору *create*, который создаёт таблицу с именем *notepad* и полями: *notekey* — ключ базы данных и *text* — строковое содержимое длиной не более 4094 символов.



```
1 -----
2 -- Сценарий создания базы данных: exampleDB.
3 -- Создается таблица notepad — личные записи студента.
4 -----
5 -- Вариант сервера:
6 -- CONNECT 'jdbc:derby://localhost:1527/exampleDB;create=true;'
7 --   USER 'upk' PASSWORD 'upkasu';
8 -----
9 -- Вариант встроенной БД:
10 CONNECT 'jdbc:derby:/home/upk/databases/exampleDB;create=true;'
11   USER 'upk' PASSWORD 'upkasu';
12 -----
13 -- Сначала удаляем все таблицы, если они были созданы.
14 drop table notepad;
15 -----
16 -- Создаем таблицу notepad
17
18 create table notepad (
19     notekey int not null,    -- ключ записи - ключ таблицы
20     text varchar(4096), -- текст записи
21     primary key (notekey)
22 );
23 -----
24 -- Вносим в таблицу некоторое количество записей:
25 insert into notepad values(321, 'Запись 1');
26 insert into notepad values(743, 'Запись 2');
27 -----
28 select * from notepad;
29 -----
30 -- Завершение работы сценария:
31 commit;
32 disconnect;
33 exit;
34 -----
```

Рисунок 2.38 — Текст сценария ~/databases/createDB.sql на языке SQL

Завершив проверку содержимого файла *createDB.sql*, можно создать целевую базу данных с помощью команды определённой выражением (2.19). В процессе выполнения этой команды в окно терминала будет выводиться протокол её исполнения.

ij createDB.sql (2.19)

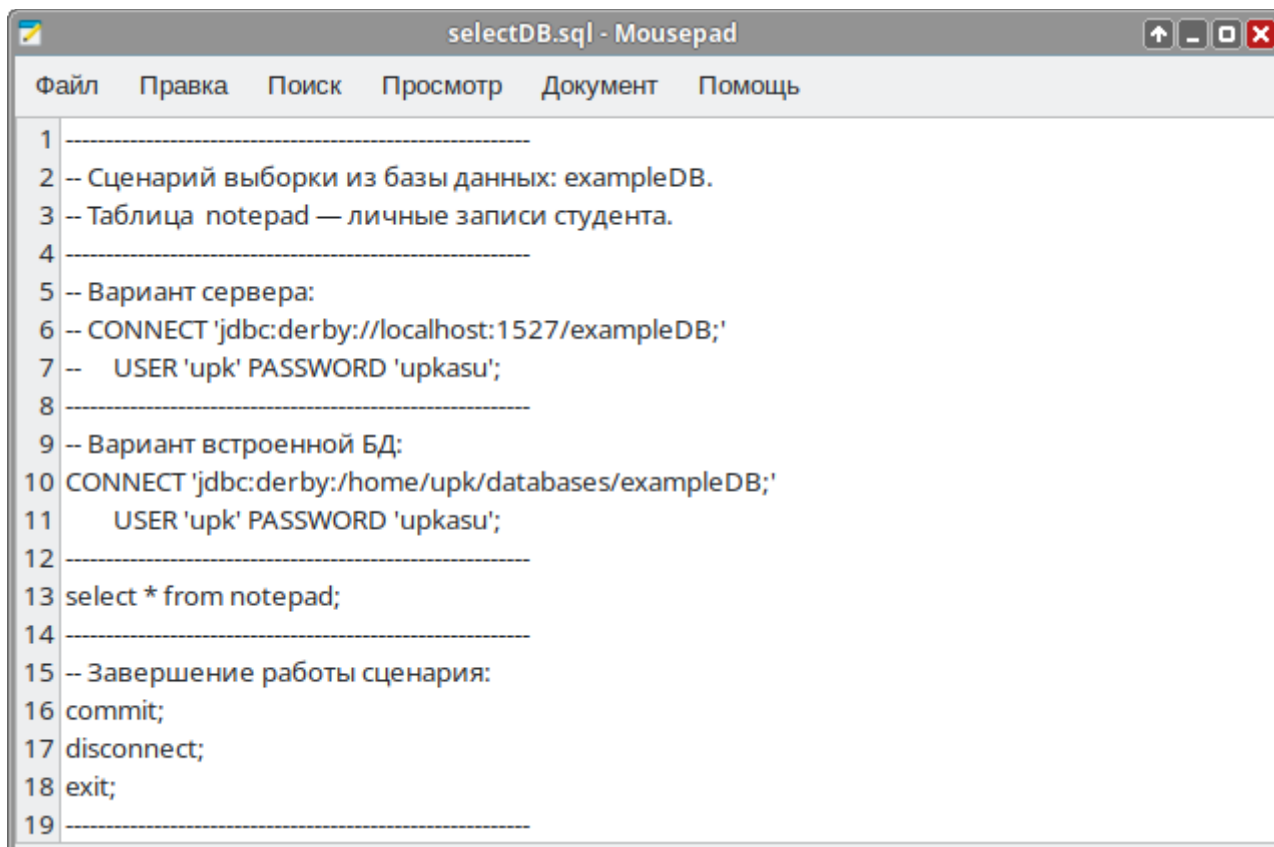
По результату выполнения команды (2.19) следует убедиться, что:

- 1) протокол сообщений сценария не содержит ошибок;
- 2) в каталоге `~/databases` должна появиться директория *exampleDB*.

Проверка правильности создания БД сценарием `~/databases/selectDB.sql`.

Создав базу данных необходимо обязательно проверить её работоспособность. Для этого предусмотрен сценарий *selectDB.sql*, текст которого представлен на рисунке 2.39.

Примечание — Сценарий *selectDB.sql* следует рассматривать как дополнительный инструмент разработчика, позволяющий быстро проверить корректность функционирования БД *exampleDB*.



```
1 -----
2 -- Сценарий выборки из базы данных: exampleDB.
3 -- Таблица notepad — личные записи студента.
4 -----
5 -- Вариант сервера:
6 -- CONNECT 'jdbc:derby://localhost:1527/exampleDB;'
7 --   USER 'upk' PASSWORD 'upkasu';
8 -----
9 -- Вариант встроенной БД:
10 CONNECT 'jdbc:derby:/home/upk/databases/exampleDB;'
11   USER 'upk' PASSWORD 'upkasu';
12 -----
13 select * from notepad;
14 -----
15 -- Завершение работы сценария:
16 commit;
17 disconnect;
18 exit;
19 -----
```

Рисунок 2.39 — Текст сценария `~/databases/selectDB.sql` на языке SQL

В тексте данного сценария следует обязательно проверить правильность задания оператора **CONNECT**, который адекватно должен соответствовать такому же оператору сценария *createDB.sql*.

Сама проверка правильности создания учебной базы данных осуществляется запуском команды, определённой выражением (2.20). На экран терминала должно быть выведено все содержимое таблицы *notepad*, без каких-либо ошибочных сообщений утилиты *ij*.

`ij selectDB.sql` (2.20)

Создание в среде Eclipse IDE проекта с именем *proj7*.

Проект *proj7* имеет важное значение для практики как этой, так и последующих лабораторных работ. В нём реализуются два примера на языке Java, которые используют не только средства пакета *java.sql*, но и внешние средства дистрибутива Apache Derby.

Создание и тестирование в проекте *proj7* класса с именем *CreateDB*.

Прикладное назначение класса *CreateDB* — демонстрационный пример создания базы данных *exampleDB* средствами языка Java, которая была бы аналогична БД создаваемой средствами сценария *createDB.sql*.

Сначала следует удалить уже созданную БД. Сделать это можно средствами файлового менеджера в окне терминала.

Затем создать в проекте *proj7* класс с именем *CreateDB*, согласно исходному тексту листинга 2.25.

Листинг 2.25 — Исходный текст класса *CreateDB* проекта *proj7*

```
package rvs.createdb;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * Пример создания БД exampleDB в каталоге ~/databases
 * @author upk
 */
public class CreateDB
{
    // Рабочие объектные переменные
    boolean flag = true;    // Флаг соединения с БД
    Connection conn;        // Объект соединения с БД

    // Конструктор класса, устанавливающий соединение с БД
    CreateDB(String dburl, String dbuser, String dbpassword)
    {

        try {
            //Подключаем необходимый драйвер
            Class.forName("org.apache.derby.jdbc.EmbeddedDriver");

            //Устанавливаем соединение с БД
            conn = DriverManager.getConnection(dburl,
                dbuser, dbpassword);

        }
        catch (ClassNotFoundException e1)
        {
            System.out.println(e1.getMessage());
            flag = false;
        }
        catch (SQLException e2)
        {
            System.out.println(e2.getMessage());
            flag = false;
        }
    }

    public static void main(String[] args)
    {
        // Объекты класса
    }
}
```

```

String url = // Адрес БД
            "jdbc:derby:/home/upk/databases/exampleDB;create=true;";
String user = "upk"; // Имя пользователя создавшего БД
String pswd = "upkasu"; // Пароль пользователя создавшего БД

// Создаем объект класса
CreateDB obj =
    new CreateDB(url, user, pswd);
if (!obj.flag)
{
    System.out.println("Не могу создать объект класса CreateDB...");
    System.exit(1);
}
else
    System.out.println("Соединение с БД exampleDB - установлено...");

try {
    // Удаляем таблицу - грубо!
    Statement s =
        obj.conn.createStatement();
    s.executeUpdate("drop table notepad");
}
catch (SQLException e1)
{
    System.out.println("e1 = " + e1.getMessage());
}

try {
    // Создаём таблицу notepad
    Statement s =
        obj.conn.createStatement();
    s.executeUpdate("CREATE TABLE notepad ("
        + "notekey int not null,"
        + "text varchar(4096),"
        + "primary key (notekey))");
    obj.conn.commit();
}
catch (SQLException e2){
    System.out.println("e2 = " + e2.getMessage());
}

try {
    // Вставляем две записи: они - изменены...
    Statement s =
        obj.conn.createStatement();
    s.executeUpdate("insert into notepad values(321, 'Запись 1-1')");
    s.executeUpdate("insert into notepad values(743, 'Запись 2-1')");
    obj.conn.commit();

    obj.conn.close();
    System.out.println("CreateDB - нормально завершила работу...");
}
catch (SQLException e3)
{
    System.out.println("e3 = " + e3.getMessage());
}
}
}

```


Примечание — Все вопросы связанные с применением SQL Derby и языка Java, применительно к Derby, можно найти в документации [~/Документы/derby-doc/docs/pdf/ref/refderby.pdf](#).

Если теперь запустить приложение **CreateDB** на исполнение, то оно завершится с ошибкой, как это показано на рисунке 2.40.

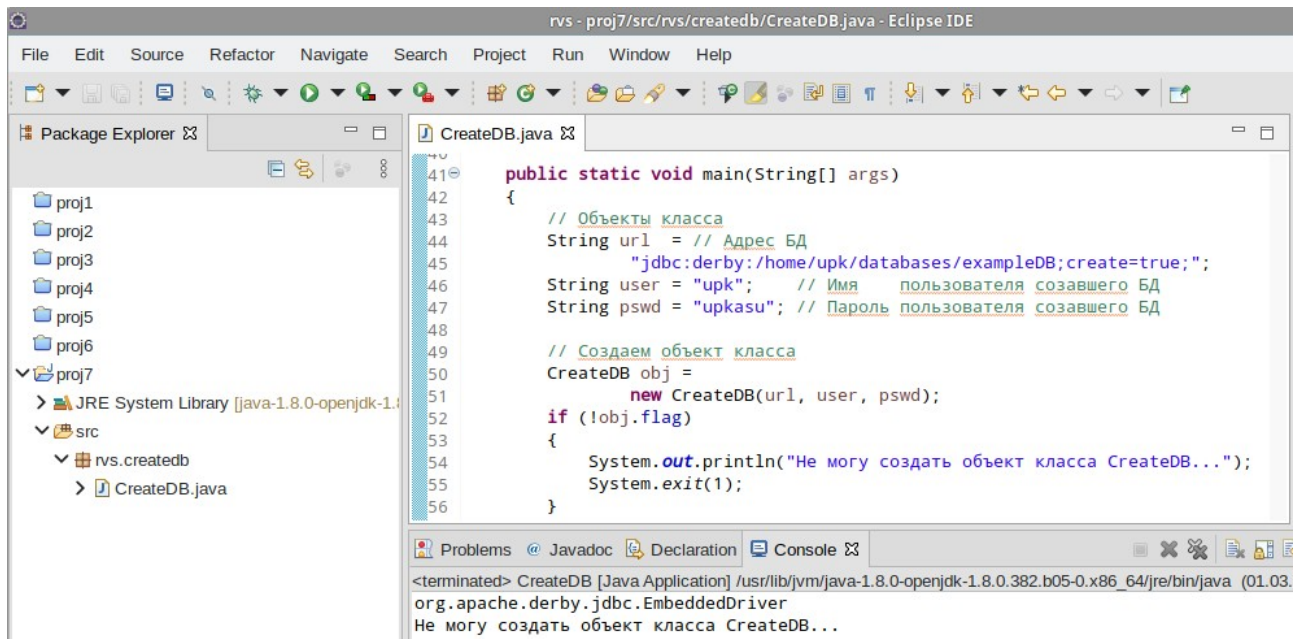


Рисунок 2.40 — Ошибка исполнения класса CreateDB из-за отсутствия драйвера СУБД

Ошибка возникает в конструкторе класса **CreateDB**, потому что к проекту не подключен драйвер СУБД Apache Derby.

Чтобы подключить к проекту нужный драйвер, следует выполнить следующие действия:

- 1) выделить мышкой в окне «**Package Explorer**» файл **CreateDB.java**;
- 2) правой кнопкой мышки активировать контекстное меню и перейти по ссылке: «**Build Path/Configure Build Path...**»; должно появиться окно настройки «**Properties for proj7**», как это показано ниже на рисунке 2.41;
- 3) в этом окне настроек проекта **proj7** следует: в левой части окна выделить пункт «**Java Build Path**», а в средней части окна выделить вкладку «**Libraries**»; затем в правой части окна активировать кнопку «**Add External JARs...**»;
- 4) в появившемся окне «**JAR Selection**» (см. рисунок 2.42) необходимо перейти в каталог **/opt/derby/lib** и выделить файл **derby.jar**, в котором и находится нужный драйвер;
- 5) после активации кнопки «**Открыть**» в окне рисунка 2.42, а затем — кнопки «**Apply and Close**» в окне рисунка 2.41, нужный драйвер будет подключен к системе и можно снова запускать класс **CreateDB** на исполнение.

После успешного завершения работы класса **CreateDB** следует перейти в окно виртуального терминала, в котором:

- 1) запустить сценарий **~/databases/selectDB.sql** на исполнение;
- 2) оформить результаты исследования в личном отчёте, сравнив результаты работы приложения **CreateDB** и сценария **createDB.sql**.

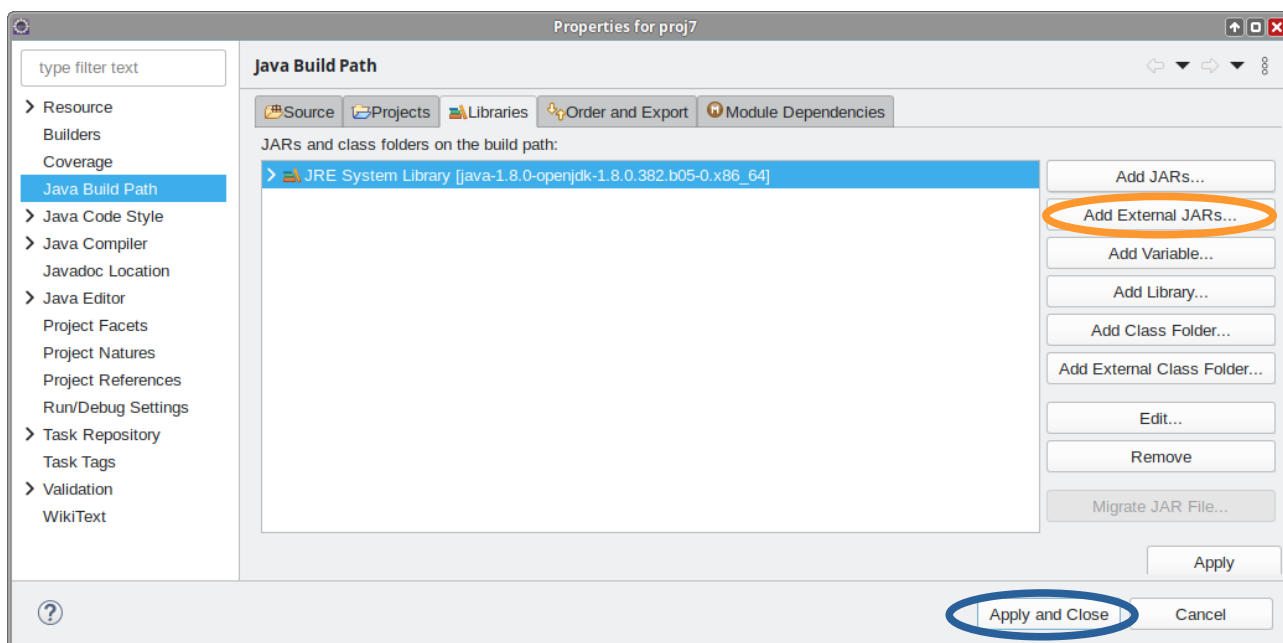


Рисунок 2.41 — Окно настроек проекта proj7

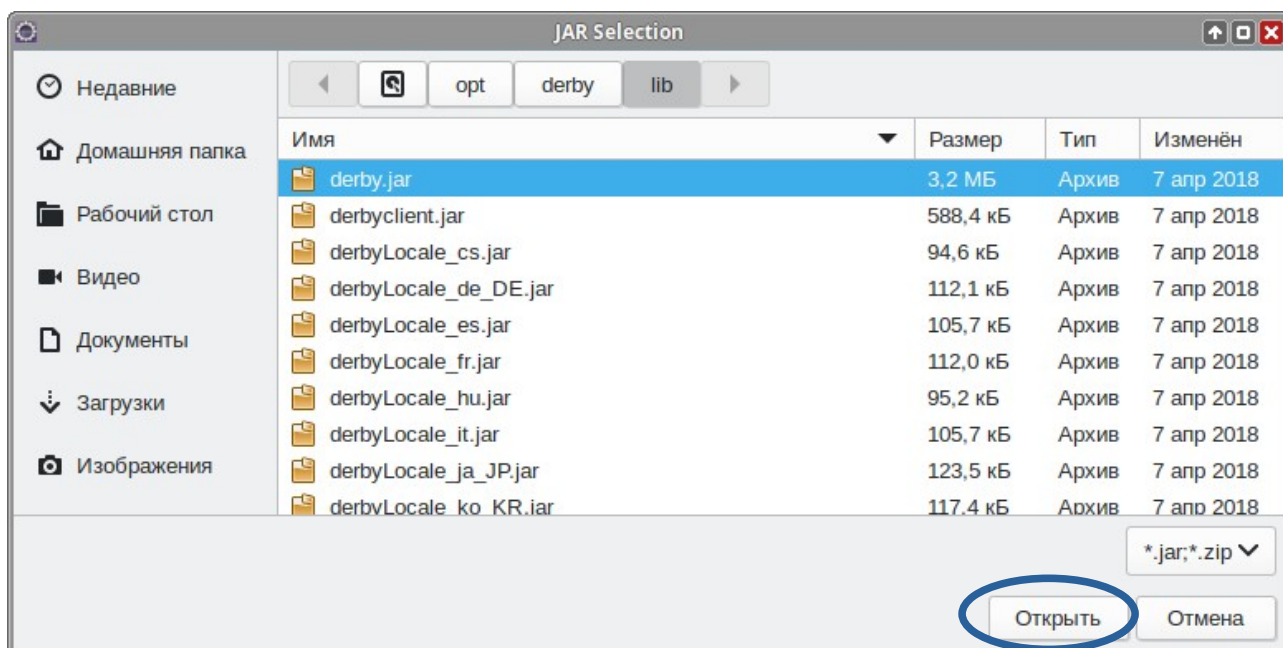


Рисунок 2.42 — Окно выбора пакета derby.jar проекта proj7

2.5.3 Реализация учебного примера Example11

Данным пунктом мы завершаем не только седьмую лабораторную работу, но и последовательное систематическое изучение базовых основ языка Java, соответствующих второму разделу учебного пособия [1].

Подводя итог изучению языка Java, создадим приложение со стандартным именем **Example11**, которое обеспечивало бы интерактивное взаимодействие пользователя с таблицей *notepad* уже созданной нами базы данных *exampleDB*.

Примечание — В целом учебный материал данной работы опирается на содержание подраздела 2.6 учебного пособия [1], но при реализации приложения *Example11* следует строго придерживаться требований именно данного пункта методических указаний.

Учебная цель данного пункта работы — получение навыков создания сосредоточенного интерактивного приложения, взаимодействующего с простейшей таблицей учебной базы данных.

Третье учебное задание:

- 1) выполнить формальное описание интерфейса сосредоточенного приложения с именем *Example11*, обеспечивающего интерактивное взаимодействие с таблицей *notepad* базы данных *exampleDB* и решающего задачи: соединение с базой данных и закрытие такого соединения, добавление записей в таблицу *notepad* и чтение всех записей этой таблицы с выводом читаемой информации на экран терминала, также интерактивный ввод данных с клавиатуры терминала;
- 2) создать в проекте *proj7* класс с именем *Example11*, согласно текста листинга 2.26, демонстрирующего технологию реализации целевого приложения языке Java;
- 3) провести тестирование реализации класса *Example11*;
- 4) результаты исследования отразить в личном отчёте.

Выполнение представленного задания рекомендуется выполнять в заданной последовательности пунктов.

Формальное описание интерфейса приложения *Example11*.

В данной работе мы не будем использовать описание интерфейса по синтаксису языка Java, изложенного в пункте 2.2.4 учебного пособия [1]. Подобное занятие оставим до лабораторных работ третьей темы, где это — действительно необходимо.

Задача данного пункта работы — описать максимально простое приложение, которое в дальнейшем будет модифицироваться на протяжении изучения всех остальных тем. Поэтому мы ограничимся табличным описанием *интерфейса методов* класса *Example11*, включая описание его конструкторов. Такое описание представлено в таблице 2.12.

Таблица 2.12 — Описание интерфейса приложения *Example11*

№	Интерфейс метода	Описание метода
1	<i>Example11(String url, String user, String passwd)</i>	Конструктор класса <i>Example11</i> со строковыми аргументами: <i>url</i> — строка соединения с базой данных <i>exampleDB</i> ; <i>user</i> — имя пользователя; <i>passwd</i> — пароль пользователя. Устанавливает соединение с базой данных, сохраняя глобальный объект типа <i>Connection</i> .
2	<i>int getResultSet()</i>	Главный метод чтения всех записей из таблицы <i>notepad</i> , сохраняя результат запроса в виде глобального объекта типа <i>ResultSet</i> . Возвращает: <i>1</i> — результат положительный; <i>0</i> — возникло исключение <i>SQLException</i> .
3	<i>int setInsert(int key, String str)</i>	Главный метод, добавляющий записи в таблицу <i>notepad</i> , имеет аргументы: <i>key</i> — целочисло (ключ записи); <i>str</i> — строка записи. Возвращает <i>-1</i> , если произошло исключение <i>SQLException</i> .
4	<i>void setClose()</i>	Главный метод, разрывающий соединение с базой данных.
5	<i>int getKey()</i>	Вспомогательный метод для ввода целого числа с клавиатуры.
6	<i>String getString()</i>	Вспомогательный метод для ввода строки с клавиатуры.
7	<i>static void main(String[] args)</i>	Вспомогательный метод, реализующий запуск и интерактивный режим работы приложения.

Более подробный анализ таблицы 2.12 будет проводиться в следующей теме дисциплины. Здесь мы отметим, что с точки зрения организации доступа к базе данных функционал класса *Example11* разделён на две группы:

- 1) конструктор класса и последующие три метода, выделенные номерами 1-4, *осуществляют взаимодействие с базой данных*; поэтому они обозначены как **главные методы**.
- 2) последние три метода, выделенные номерами 5-7, *осуществляют локальную обработку данных*, включая взаимодействие с конечным пользователем; поэтому они обозначены как **вспомогательные методы**.

Примечание — Поскольку сейчас мы реализуем *сосредоточенное приложение*, то разделение методов на главные и вспомогательные не играет никакой существенной роли, но когда мы будем реализовывать *распределённую систему*, это разделение становится существенным.

Реализация и тестирование приложения Example11.

В проекте *proj7* системы Eclipse IDE нужно создать класс *Example11* в соответствии с содержимым листинга 2.6. Исходный текст этого листинга представляет конкретную реализацию целевого приложения, соответствующую заявленному и представленному в таблице 2.12 описанию интерфейса.

Примечание — Не следует в работе использовать исходный текст класса Example11, представленный в учебном пособии [1].

Исходный текст листинга 2.6 хорошо комментирован, поэтому не требует более подробного его описания.

Студенту необходимо:

- 1) внимательно изучить содержимое текста класса *Example11*;
- 2) запустить этот класс на исполнение;
- 3) протестировать работу этого класса и результаты работы отобразить в личном отчёте.

Листинг 2.26 — Исходный текст класса Example11 проекта proj7

```
package ru.tusur.asu;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class Example11 {

    /**
     * Глобальные объектные переменные класса
     */
    Connection conn;           // Объект соединения
    boolean flag = true;       // Флаг наличия соединения

    ResultSet rs;              // Результат запроса SELECT
```

```

/**
 * Главные методы класса
 */
// Конструктор
Example11(String dburl, String dbuser, String dbpassword)
{

    try {
        // Подключаем необходимый драйвер
        Class.forName("org.apache.derby.jdbc.EmbeddedDriver");

        // Устанавливаем соединение с БД
        conn = DriverManager.getConnection(dburl,
            dbuser, dbpassword);

    }
    catch (ClassNotFoundException e1)
    {
        System.out.println(e1.getMessage());
        flag = false;
    }
    catch (SQLException e2)
    {
        System.out.println(e2.getMessage());
        flag = false;
    }
}

// Метод, реализующий SELECT
public int getResultSet()
{
    String sql1 = "SELECT * FROM notepad ORDER BY notekey";

    try
    {
        Statement st =
            conn.createStatement();
        rs = st.executeQuery(sql1);
        return 1;

    } catch (SQLException e2){
        System.out.println(e2.getMessage());
        return 0;
    }
}

// Метод, реализующий INSERT
public int setInsert(int key, String str)
{
    String sql2 = "INSERT INTO notepad values( ? , ? )";

    try
    {
        PreparedStatement pst =
            conn.prepareStatement(sql2);

        // Установка первого параметра
        pst.setInt(1, key);
        // Установка второго параметра

```

```

        pst.setString(2, str);

        return pst.executeUpdate();
    } catch (SQLException e2){
        System.out.println(e2.getMessage());
        return -1;
    }
}

// Закрытие соединения
public void setClose()
{
    try
    {
        rs.close();
        conn.commit();
        conn.close();

    } catch (SQLException e2){
        System.out.println(e2.getMessage());
    }
}

/**
 * Второстепенные методы класса
 */
// Метод чтения целого числа со стандартного ввода
public int getKey()
{
    int ch1 = '0';
    int ch2 = '9';
    int ch;
    String s = "";

    try
    {
        while(System.in.available() == 0) ;

        while(System.in.available() > 0)
        {
            ch = System.in.read();
            if (ch == 13 || ch < ch1 || ch > ch2)
                continue;
            if (ch == 10)
                break;

            s += (char)ch;
        };
        if (s.length() <= 0)
            return -1;
        ch = new Integer(s).intValue();
        return ch;

    } catch (IOException e1){
        System.out.println(e1.getMessage());
        return -1;
    }
}

```

```

// Метод чтения строки текста со стандартного ввода
public String getString()
{
    String s = "\r\n";
    String text = "";
    int n;
    char ch;
    byte b[];

    try
    {
        // Ожидаем поток ввода
        while(System.in.available() == 0) ;
        s = "";
        while((n = System.in.available()) > 0)
        {
            b = new byte[n];
            System.in.read(b);
            s += new String(b);
        };
        // Удаляем последние символы '\n' и '\r'
        n = s.length();
        while (n > 0)
        {
            ch = s.charAt(n-1);
            if (ch == '\n' || ch == '\r')
                n--;
            else
                break;
        }
        // Выделяем подстроку
        if (n > 0)
            text = s.substring(0, n);
        else
            text = "";
        return text;
    } catch (IOException e1){
        System.out.println(e1.getMessage());
        return "Ошибка...";
    }
}

// Запуск и интерактивный режим исполнения
public static void main(String[] args)
{
    System.out.print("Программа ведения записей в БД exampleDB.\n"
        + "Используются методы класса Example11:\n"
        + "\t1) если ключ - пустой, то завершаем программу;\n"
        + "\t2) если текст - не пустой, то добавляем его.\n"
        + "Нажми Enter - для продолжения ...\n"
        + "-----");

    /**
     * Исходные данные: адрес, имя и пароль
     */
    String url = "jdbc:derby:/home/upk/databases/exampleDB";
    String user = "upk";
    String password = "upkasu";
}

```

```

// Создаём объект класса
Example11 obj =
    new Example11(url, user, password);
// Ждём, пока пользователь читает заголовок сообщения...
obj.getKey();

// Проверяем наличие соединения с базой данных
if (!obj.flag)
{
    System.out.println("Не могу создать объект класса Example11...");
    // Аварийно завершаем работу программы
    System.exit(1);
}

// Рабочие переменные
int ns;                // Число прочитанных строк
int nb;                // Число прочитанных байт
String text, s;        // Строка введенного текста

// Цикл обработки запросов
while(obj.flag)
{
    // Печатаем заголовок ответа
    System.out.println("        Ключ    Текст\n"
        + "-----");
    obj.getResultSet();

    if (obj.rs == null)
    {
        System.out.println("Отсутствует результат SELECT...");
        break;
    }
    ns = 0;
    try
    {
        // Выводим (построчно) результат запроса к БД
        while(obj.rs.next())
        {
            System.out.format("%10d", obj.rs.getInt(1));
            System.out.println("    " + obj.rs.getString(2));
            ns++;
        }
        // Выводим итог запроса
        System.out.println(
            "-----\n"
            + "Прочитано " + ns + " строк(и)\n"
            + "-----\n"
            + "Формируем новый запрос!");
        /**
         * Интерактивное формирование запроса к базе данных
         */
        System.out.print("\nВведи ключ или Enter: ");
        nb = obj.getKey();

        if (nb == -1)
            break;    // Завершаем работу программы

        System.out.print("Строка текста или Enter: ");
        s = obj.getString();
        text = s;
    }
}

```

```

        while (s.length() > 0)
        {
            System.out.print("Строка текста или Enter: ");
            s = obj.getString();
            if (s.length() > 0)
                text += ("\n" + s);
        }

        if (text.length() <= 0)
            text = "Нет данных...";

        ns = obj.setInsert(nb, text);
        if (ns == -1)
            System.out.println("\nОшибка добавления строки !!!");
        else
            System.out.println("\nДобавлено " + ns + " строк...");

    } catch (SQLException e1){
        System.out.println(e1.getMessage());
        break;
    }
}

/**
 * Закрываем все объекты и разрываем соединение
 */
obj.setClose();
System.out.println("Программа завершила работу...");
}
}

```

Завершив работы по данному пункту, следует подвести общий итог и завершить выполнение лабораторной работы №7.

Примечание — Данная лабораторная работа завершает практическое освоение студентом учебного материала второго раздела учебного пособия [1].

3 ОБЪЕКТНЫЕ РАСПРЕДЕЛЁННЫЕ СИСТЕМЫ

3.1 Лабораторная работа №8.

Программное проектирование распределённой системы

3.2 Лабораторная работа №9.

Реализация сервера PBC по технологии RMI

3.3 Лабораторная работа №10.

Реализация клиента PBC по технологии RMI

4 WEB-ТЕХНОЛОГИИ РАСПРЕДЕЛЁННЫХ СИСТЕМ

4.1 Лабораторная работа №11.

Технология сервлетов на базе сервера Apache Tomcat

4.2 Лабораторная работа №12.

Технология JSP для формирования динамических HTML-страниц

4.3 Лабораторная работа №13.

Шаблон проектирования MVC

5 СЕРВИС-ОРИЕНТИРОВАННЫЕ АРХИТЕКТУРЫ

5.1 Лабораторная работа №14. Проектирование трёхзвенной РВС в стиле REST

5.2 Лабораторная работа №15. Реализация модели приложения в стиле REST

5.3 Лабораторная работа №16. Реализация распределённого приложения в стиле REST

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Резник, В.Г. Распределенные вычислительные сети: Учебное пособие [Электронный ресурс] / В.Г. Резник. — Томск: ТУСУР, 2019. — 211 с. — Режим доступа: <https://edu.tusur.ru/publications/9072>.
2. Резник, В.Г. Распределенные вычислительные системы: Практические занятия по направлению подготовки бакалавриата 09.03.01 [Электронный ресурс] / В.Г. Резник. — Томск: ТУСУР, 2019. — 96 с. — Режим доступа: <https://edu.tusur.ru/publications/9108>.
3. Шилдт, Герберт. Java 8. Полное руководство; 9-е изд.: Пер. с англ. - М.: ООО "И.Д. Вильямс", 2015. —1376 с.
4. Резник, В.Г. Учебный программный комплекс кафедры АСУ на базе ОС ArchLinux: Учебно-методическое пособие для студентов направления 09.03.01, Направление подготовки "Программное обеспечение средств вычислительной техники и автоматизированных систем" [Электронный ресурс] / В.Г. Резник. — Томск: ТУСУР, 2016. — 33 с. — Режим доступа: <https://edu.tusur.ru/publications/6238>.
5. Java DB Reference Manual (Документация по СУБД Derby) Дополнительная документация по дисциплине в файле: refderby.pdf.