Лабораторная работа №3
Построение клиент-серверной архитектуры на базе протокола TCP/IP

Содержание

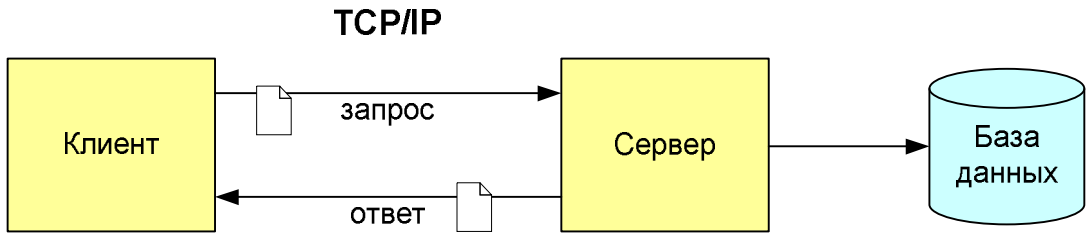
1. Лабораторная работа 3

1.1. Постановка задачи

На основе программы, разработанной в рамках лабораторной работы №2, создать серверную программу, обеспечивающую выполнение удаленных запросов по управлению объектами. Сведения об объектах должны храниться в базе данных.

Разработать клиентское приложение, отправляющее серверу запросы на ввод, редактирование и получение информации об объектах.

Взаимодействие между клиентом и сервером должно осуществляться по протоколу TCP/IP.



1.2. Требования к работе

Сервер запускается в фоновом режиме и не имеет пользовательского интерфейса. Сервер должен обеспечивать выполнение операций, представленных в вашем варианте задания.

На каждый запрос клиента сервер должен отправлять ответ. Ответы сервера должны различаться в зависимости от результата выполнения запроса клиента (положительный/отрицательный ответ).

Сервер ориентирован на взаимодействие с единственным клиентом. Поддержку взаимодействия сервера с несколькими клиентами реализовывать не требуется.

Сервер должен обеспечивать уникальность идентификаторов объектов при выполнении операций добавления и редактирования.

Клиентское приложение отправляет запросы серверу и демонстрирует ответы пользователю. Клиентское приложение не требует создания пользовательского интерфейса. Тестирование работоспособности клиента осуществляется на основе сценариев, демонстрирующих возможности программы.

Формат информационных сообщений, которыми обмениваются клиент и сервер, определяется в соответствии с номером варианта (см. раздел 2.3).

Рекомендуемый язык программирования – Java.

1.3. Требования к оформлению отчета

Отчет должен содержать:

- титульный лист
- постановку задачи
- исходный код программ
- описание программ (*описание классов, методов, полей*)
- описание протокола взаимодействия клиента и сервера (*то есть ПОДРОБНОЕ описание формата и структуры ВСЕХ типов информационных сообщений, которыми обмениваются клиент и сервер между собой, по аналогии с табл. 1 и 2*)

1.4. Варианты заданий

Вариант 1	
Предметная область	Карта мира
Объекты	Страны, Города
Примечание	Карта мира содержит множество *стран*. Для каждой *страны* определено множество *городов*.
Требуемые операции	1. Добавление новой страны 2. Удаление страны 3. Добавление нового города в заданную страну 4. Удаление города 5. Редактирование города 6. Подсчет количества городов в стране 7. Выдача полного списка городов с указанием названия страны 8. Выдача списка городов для заданной страны 9. Выдача полного списка стран
Формат сообщений	Строка с разделителем

Вариант 2	
Предметная область	Библиотека
Объекты	Авторы, Книги
Примечание	Книги в библиотеке сгруппированы по *авторам*. У каждого *автора* имеется множество *книг*.
Требуемые операции	1. Добавление нового автора 2. Удаление автора 3. Добавление новой книги для автора 4. Удаление книги 5. Редактирование книги 6. Подсчет общего количества книг 7. Выдача полного списка книг с указанием ФИО автора 8. Выдача книг заданного автора 9. Выдача списка авторов
Формат сообщений	Двоичный

Вариант 3	
Предметная область	Отдел кадров
Объекты	Подразделения, Сотрудники
Примечание	Имеется множество *подразделений* предприятия. В каждом *подразделении* работает множество *сотрудников*.
Требуемые операции	1. Добавление нового подразделения 2. Удаление подразделения 3. Прием на работу сотрудника в заданное подразделение 4. Увольнение сотрудника 5. Редактирование личных данных сотрудника 6. Перевод сотрудника из одного подразделения в другое 7. Подсчет количества сотрудников в подразделении 8. Выдача списка сотрудников для заданного подразделения 9. Выдача списка подразделений
Формат сообщений	Строка с разделителем

Вариант 4	
Предметная область	Учебный отдел
Объекты	Группы, Студенты
Примечание	Имеется множество учебных *групп*. Каждая группа включает в себя множество *студентов*.
Требуемые операции	1. Добавление новой группы 2. Удаление группы 3. Зачисление студента в заданную группу 4. Отчисление студента 5. Перевод студента в заданную группу 6. Редактирование личных данных студента 7. Выдача списка студентов для заданной группы 8. Выдача списка групп 9. Выдача полного списка студентов с указанием названия группы
Формат сообщений	Двоичный

Вариант 5	
Предметная область	Автосалон
Объекты	Производители автомобилей, Марки
Примечание	*Марки* автомобилей сгруппированы по производителям. У каждого *производителя* имеется множество *марок*.
Требуемые операции	1. Добавление производителя 2. Удаление производителя 3. Добавление новой марки заданного производителя 4. Удаление марки 5. Редактирование марки 6. Подсчет количества марок у производителя 7. Выдача полного списка марок с названием производителя 8. Выдача полного списка производителей 9. Выдача списка марок заданного производителя
Формат сообщений	Строка с разделителем

Вариант 6	
Предметная область	Агентство новостей
Объекты	Категории новостей, Новости
Примечание	Новости сгруппированы по *категориям*. У каждой *категории* имеется множество *новостей*.
Требуемые операции	1. Добавление новой категории 2. Удаление категории 3. Добавление новости заданной категории 4. Удаление новости 5. Редактирование новости 6. Подсчет количества новостей в категории 7. Выдача новости по идентификатору 8. Выдача полного списка новостей для заданной категории 9. Выдача полного списка категорий
Формат сообщений	Двоичный

Вариант 7	
Предметная область	Продуктовый магазин
Объекты	Категория продукта, Продукт
Примечание	*Продукты* в магазине сгруппированы по *категориям*. Для каждой *категории* определено множество *продуктов*.
Требуемые операции	1. Добавление новой категории 2. Удаление категории 3. Добавление продукта заданной категории 4. Удаление продукта 5. Редактирование продукта 6. Подсчет количества продуктов в категории 7. Поиск продуктов по названию 8. Выдача списка продуктов заданной категории 9. Выдача списка категорий
Формат сообщений	Строка с разделителем

Вариант 8	
Предметная область	Футбол
Объекты	Команды, Игроки
Примечание	Имеется множество футбольных *команд*. Для каждой *команды* определено множество *игроков*.
Требуемые операции	1. Добавление новой команды 2. Удаление команды 3. Добавление нового игрока в команду 4. Удаление игрока 5. Редактирование сведений об игроке 6. Перевод игрока из одной команды в другую 7. Выдача полного списка игроков с указанием названия команды 8. Выдача полного списка команд 9. Выдача списка игроков заданной команды
Формат сообщений	Двоичный

Вариант 9	
Предметная область	Музыкальный магазин
Объекты	Исполнители, Альбомы
Примечание	В музыкальном магазине *альбомы* сгруппированы по *исполнителям*. Для каждого *исполнителя* задано множество *альбомов*.
Требуемые операции	1. Добавление нового исполнителя 2. Удаление исполнителя 3. Добавление нового альбома заданного исполнителя 4. Удаление альбома 5. Редактирование данных об исполнителе 6. Подсчет количества альбомов исполнителя 7. Выдача полного списка альбомов с указанием исполнителя 8. Выдача списка альбомов заданного исполнителя 9. Выдача полного списка исполнителей
Формат сообщений	Строка с разделителем

Вариант 10	
Предметная область	Аэропорт
Объекты	Авиакомпании, Рейсы
Примечание	Имеется множество *авиакомпаний*. Для каждой *авиакомпании* определены ее *рейсы*.
Требуемые операции	1. Добавление новой компании 2. Удаление компании 3. Добавление нового рейса 4. Отмена (удаление) рейса 5. Редактирование рейса 6. Поиск рейса по номеру рейса (идентификатору) 7. Выдача полного списка рейсов с указанием названия авиакомпании 8. Выдача полного списка авиакомпаний 9. Выдача списка рейсов заданной авиакомпании
Формат сообщений	Двоичный

Вариант 11	
Предметная область	Файловая система
Объекты	Папки, Файлы
Примечание	Имеется множество *папок* (независимых друг от друга). Для каждой *папки* определено множество *файлов*.
Требуемые операции	1. Добавление новой папки 2. Удаление папки 3. Добавление нового файла в заданную папку 4. Удаление файла 5. Редактирование файла 6. Перенос файла в заданную папку 7. Копирование файла в заданную папку 8. Выдача полного списка папок 9. Выдача списка файлов для заданной папки
Формат сообщений	Строка с разделителем

Вариант 12	
Предметная область	Расписание занятий
Объекты	Дни недели, Занятия
Примечание	Имеется множество *дней*. Для каждого *дня* определен перечень *занятий*.
Требуемые операции	1. Добавление нового дня 2. Удаление дня 3. Добавление нового занятия 4. Удаление занятия 5. Редактирование занятия 6. Запрос количества занятий в заданный день 7. Выдача полного списка занятий с указанием дня 8. Выдача списка занятий для заданного дня 9. Выдача списка дней
Формат сообщений	Двоичный

Вариант 13	
Предметная область	Записная книжка
Объекты	Календарные дни, Мероприятия
Примечание	Имеется множество *дней*. Для каждого *дня* определен перечень *мероприятий*.
Требуемые операции	1. Добавление нового дня 2. Удаление дня 3. Добавление нового мероприятия 4. Удаление мероприятия 5. Перенос заданного мероприятия на заданный день 6. Запрос количества мероприятий в заданный день 7. Выдача полного списка мероприятий с указанием дня 8. Выдача полного списка дней 9. Выдача списка мероприятий для заданного дня
Формат сообщений	Строка с разделителем

Вариант 14	
Предметная область	Видеомагазин
Объекты	Жанры, Фильмы
Примечание	Имеется множество *жаров*. Для каждого *жанра* определен перечень *фильмов*.
Требуемые операции	1. Добавление нового жанра 2. Удаление жанра 3. Добавление нового фильма 4. Удаление фильма 5. Редактирование фильма 6. Запрос количества фильмов заданного жанра 7. Поиск фильма по названию 8. Выдача списка фильмов заданного жанра 9. Выдача полного списка жанров
Формат сообщений	Двоичный

Вариант 15	
Предметная область	Железная дорога
Объекты	Дороги, Станции
Примечание	Имеется множество *железных дорог*. В ведомстве каждой дороги находится множество *станций*.
Требуемые операции	1. Добавление новой дороги 2. Удаление дороги 3. Добавление новой станции 4. Удаление станции 5. Редактирование станции 6. Запрос количества станции на заданной дороге 7. Поиск станции по названию 8. Выдача полного списка дорог 9. Выдача списка станций для заданной дороги
Формат сообщений	Строка с разделителем

Вариант 16	
Предметная область	Склад
Объекты	Секции, Товары
Примечание	*Товары* на складе сгруппированы по *секциям*. Для каждой *секции* задано множество *товаров*.
Требуемые операции	1. Добавление новой секции 2. Удаление секции 3. Добавление нового товара в заданную секцию 4. Удаление товара 5. Редактирование товара 6. Поиск товара по названию 7. Запрос количества товаров в заданной секции 8. Выдача полного списка секций 9. Выдача списка товаров для заданной секции
Формат сообщений	Двоичный

Вариант 17	
Предметная область	Кафедра университета
Объекты	Преподаватели, Дисциплины
Примечание	На кафедре имеется множество *преподавателей*. Для каждого *преподавателя* задано множество *дисциплин*.
Требуемые операции	1. Прием на работу (добавление) нового преподавателя 2. Увольнение (удаление) преподавателя 3. Добавление новой дисциплины 4. Удаление дисциплины 5. Редактирование личных данных преподавателя 6. Запрос количества дисциплин у преподавателя 7. Поиск дисциплины по названию 8. Выдача полного списка преподавателей 9. Выдача списка дисциплин для заданного преподавателя
Формат сообщений	Строка с разделителем

Вариант 18	
Предметная область	Программное обеспечение
Объекты	Производители, Программные продукты
Примечание	Программные *продукты* сгруппированы по *производителям*. Для каждого *производителя* задано множество *продуктов*.
Требуемые операции	1. Добавление нового производителя 2. Удаление производителя 3. Добавление нового продукта 4. Удаление продукта 5. Редактирование продукта 6. Поиск продукта по названию 7. Запрос количества продуктов у производителя 8. Выдача полного списка производителей 9. Выдача списка продуктов заданного производителя
Формат сообщений	Двоичный

Вариант 19	
Предметная область	Геометрия
Объекты	Многоугольники, Вершины
Примечание	Имеется множество *многоугольников*. Каждый *многоугольник* состоит из произвольного числа *вершин*.
Требуемые операции	1. Добавление нового многоугольника 2. Удаление многоугольника 3. Добавление новой вершины 4. Удаление вершины 5. Редактирование координат вершины 6. Запрос количества вершин у многоугольника 7. Выдача полного списка многоугольников 8. Выдача полного списка вершин заданного многоугольника 9. Сравнение двух заданных многоугольников на равенство
Формат сообщений	Строка с разделителем

Вариант 20	
Предметная область	Схема метро
Объекты	Линии, Станции
Примечание	Имеется множество *линий* метрополитена. Каждая *линия* состоит из последовательности *станций*.
Требуемые операции	1. Добавление новой линии 2. Удаление линии 3. Добавление новой станции 4. Удаление станции 5. Редактирование станции 6. Поиск станции по названию 7. Запрос количества станций на линии 8. Выдача списка станций для заданной линии. 9. Выдача полного списка линий
Формат сообщений	Двоичный

2. Рекомендации по выполнению работы

2.1. Взаимодействие по протоколу TCP/IP

Для взаимодействия по протоколу TCP/IP в языке Java можно использовать интерфейс *сокетов*. Термин "сокет" (socket) обозначает одновременно библиотеку сетевых интерфейсов и оконечное устройство канала связи (точку связи), через которое процесс может передавать или получать данные.

TCP/IP-сокеты позволяют реализовать надежные двунаправленные, ориентированные на работу с потоками соединения точка-точка между удаленными узлами.

В целом алгоритм работы системы клиент-сервер выглядит следующим образом:

1. Сервер подключается к порту на хосте и ждет соединения с клиентом
2. Клиент создает сокет и пытается соединиться через него с удаленным сервером
3. В случае установки соединения, сервер получает сокет для взаимодействия с клиентом и переходит в режим ожидания команд от клиента
4. Клиент формирует команду и передает ее через сокет серверу, после чего переходит в режим ожидания ответа
5. Сервер принимает команду через сокет, выполняет ее и пересылает ответ клиенту, после чего вновь переходит в режим ожидания запросов от клиента

Итак, на стороне сервера необходимо начать прослушивать определенный порт TCP/IP, для того чтобы удаленный клиент мог выполнять подключения. Это можно сделать при помощи класса java.net.ServerSocket.

Первоначально, необходимо создать экземпляр класса ServerSocket и указать в конструкторе порт, который будет прослушиваться. Если заданный порт занят, будет сгенерировано исключение IOException.

```
ServerSocket server = null;
try
{
    server = new ServerSocket(12345);
}
catch(IOException e) {}
```

Далее, необходимо перевести сервер в режим ожидания подключений. Делается это при помощи метода accept() класса ServerSocket. Метод accept выполняет ожидание соединения со стороны клиента и возвращает объект класса java.net.Socket в случае успешной работы. В случае ошибки, метод генерирует исключение IOException.

```
Socket sock = null;
try
{
    System.out.println("Waiting for a client...");
    sock = server.accept();
    System.out.println("Client connected");
}
catch(IOException e) {}
```

При необходимости можно установить максимальный интервал ожидания клиентских подключений. Делается это при помощи метода setSoTimeout класса ServerSocket.

Полученный объект java.net.Socket далее будет использоваться для приема и передачи данных клиенту.

На стороне клиента необходимо осуществить подключение к серверу. Для этого используется класс java.net.Socket. На клиенте необходимо явно создать объект класса Socket и передать в конструктор параметры подключения к серверу: IP-адрес сервера и порт. В случае ошибки, конструктор сгенерирует исключение IOException.

```
Socket sock = null;
try
{
    System.out.println("Connecting to server...");
    sock = new Socket("localhost",12345);
    System.out.println("Connected");
}
catch(IOException e) {}
```

Созданный объект java.net.Socket далее будет использоваться для приема и передачи данных серверу.

2.2. Прием и передача данных

Обмен данными между клиентом и сервером осуществляется через объекты java.net.Socket. После того, как соединение между клиентом и сервером установлено и обе программы получили свои объекты Socket, они могут начинать общаться.

Чтобы передать информацию удаленной программе, необходимо осуществить запись этой информации в поток вывода сокета. Получить поток вывода можно при помощи метода getOutputStream объекта класса Socket.

Чтобы получить информацию, переданную со стороны удаленной программы, необходимо выполнить чтение из потока ввода сокета. Поток ввода возвращается при помощи метода getInputStream объекта класса Socket.

2.2.1 Прием и передача текстовых данных

Для приема/передачи текстовых данных необходимо сконструировать потоки BufferedReader и PrintWriter на основе потоков ввода-вывода сокета (getInputStream() и getOutputStream()).

В следующем примере мы передадим с клиента на сервер фразу «Hello, server».

```
try
{
    PrintWriter out = new PrintWriter(sock.getOutputStream(), true);
    out.println("Hello, server!");
} catch (IOException e) {}
```

Теперь разработаем серверный код, который получит данное сообщение и выведет его на экран.

```
try
{
    BufferedReader in = new BufferedReader(
                    new InputStreamReader(sock.getInputStream( )));
    String messageFromClient = in.readLine();
    System.out.println(messageFromClient);
} catch (IOException e) {}
```

При чтении из потока, сокет начинает ожидать поступления данных от удаленного партнера. Установить маскимальный интервал ожидания можно при помощи метода setSoTimeout класса Socket.

2.2.2 Прием и передача двоичных данных

Для приема/передачи двоичных данных необходимо сконструировать потоки DataInputStream и DataOutputStream на основе потоков ввода-вывода сокета (getInputStream() и getOutputStream()).

В следующем примере мы передадим с клиента на сервер массив целых чисел:

```
// Создаю массив
int N = 3;
int A[] = new int[3];
for (int i=0; i<N; i++)
    A[i] = 10*i;
// Передаю массив на сервер
try
{
    DataOutputStream out =
            new DataOutputStream(sock.getOutputStream());
    // Передаю количество элементов
    out.writeInt(N);
    // Передаю элементы
    for (int i=0; i<N; i++)
        out.writeInt(A[i]);
} catch (IOException e) {}
```

Теперь разработаем серверный код, который получит данные числа и выведет их на экран.

```
try
{
    DataInputStream in = new DataInputStream(sock.getInputStream());
    // Получаю количество чисел
    int N = in.readInt();
    // Получаю числа и вывожу на экран
    for (int i=0; i<N; i++)
    {
        int val = in.readInt();
        System.out.println(val);
    }

} catch (IOException e) {}
```

2.3. Форматы данных

Формат информационных сообщений, которыми обмениваются клиент и сервер, определяется в соответствии с номером варианта.

В заданиях предлагается использовать следующие форматы сообщений:

- двоичный;
- строка с разделителем.

Рассмотрим различные способы представления и передачи данных на примере информационного сообщения, содержащего сведения о результатах футбольного матча. Пусть результат матча характеризуется названиями команд и количеством голов, забитых каждой командой:

Результат футбольного матча

Название команды 1	Название команды 2	Голы команды 1	Голы команды 2

2.3.1. Двоичный формат

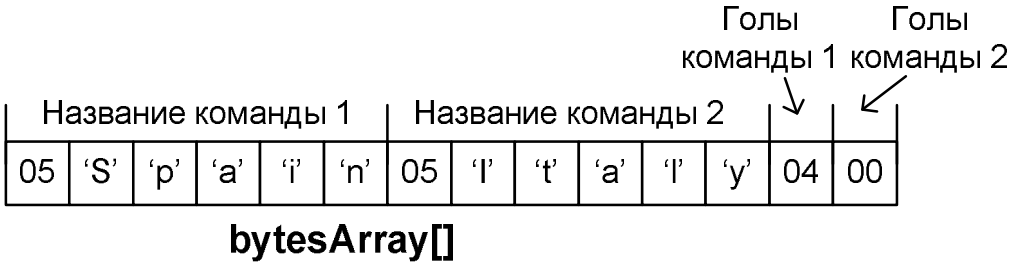
При использовании двоичного формата возможны следующие варианты передачи информации:

1. Множество отдельных пакетов определенного типа данных (данный способ представлен в примере 2.4.1). Каждый передаваемый пакет содержит сведения об одном параметре. Например:

```
DataOutputStream out =
            new DataOutputStream(sock.getOutputStream());
out.writeUtf("Spain");
out.writeUtf("Italy");
out.writeInt(4);
out.writeInt(0);
```

2. Массив байт, включающий в себя все поля информационного сообщения

```
DataOutputStream out =
              new DataOutputStream(sock.getOutputStream());
byte[] byteArray = new byte[...];
...
out.write(byteArray);
```



3. Сериализованный объект:

```
public class Match implements Serializable
{   public String command1;
    public String command2;
    public int    score1;
    public int    score2;
}
...
Match m = new Match();
m.command1 = "Spain"; m.command2 = "Italy";
m.score1 = 4; m.score2 = 0;
ObjectOutputStream out =
              new ObjectOutputStream(sock.getOutputStream());
out.writeObject(m);
```

2.3.2. Строка с разделителем

При использовании данного формата, все поля информационного сообщения записываются в текстовом виде в одну строку, разделенные некоторым служебным символом.

Например, в следующей строке в качестве разделителя используется символ процента: `"Spain%Italy%4%0"`.

```
PrintWriter out = new PrintWriter(sock.getOutputStream(), true);
out.println("Spain%Italy%4%0");
```

Передача сообщений в формате строки с разделителем представлена в примере 2.4.2.

2.4. Взаимодействие в режиме клиент-сервер

При взаимодействии по схеме «клиент-сервер» инициатором взаимодействия выступает клиент. Таким образом, сервер всегда должен находиться в режиме ожидания, так как сервер не знает, в какой момент времени от клиента поступит очередной запрос.

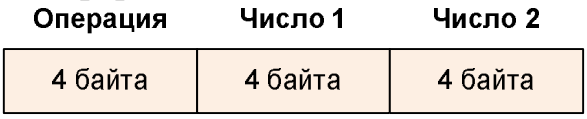
Клиент отправляет серверу запрос на выполнение определенной операции, и переходит в режим ожидания ответа. Сервер, получает запрос, выполняет необходимую операцию и отправляет клиенту ответ. Клиент обрабатывает ответ и продолжает свое нормально функционирование. Сервер же, вновь переходит в режим ожидания запросов.

2.4.1. Пример 1. Передача данных в двоичном формате

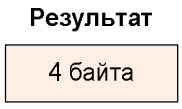
Следующий простой пример демонстрирует взаимодействие в режиме клиент-сервер через сокеты.

Разработаем сервер, выполняющий арифметические операции над заданными числами, и клиента, взаимодействующего с эти сервером. Сервер принимает от клиента два числа и код арифметической операции (будем считать, что 0 – это сложение, 1 – вычитание), выполняет над числами необходимое действие и возвращает клиенту результат. Данные, передаваемые между клиентом и сервером, представлены в двоичном формате.

Клиент передается запрос:



Сервер возвращает ответ:



Серверный код:

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

public class Server {
    private ServerSocket server = null;
    private Socket sock = null;
    private DataOutputStream out = null;
    private DataInputStream in = null;

    // запустить сервер
    public void start(int port) throws IOException
    {
        server = new ServerSocket(port);
        while (true)
        {
            // Принимаем соединение от нового клиента
            sock = server.accept();
            // Получаем потоки ввода-вывода
            in = new DataInputStream(sock.getInputStream( ));
            out = new DataOutputStream(sock.getOutputStream());
            // Пока соединение активно, обрабатываем запросы
            while (processQuery());
        }
    }

    // обработка запроса
    private boolean processQuery()
    {
        try
        {
            // Получаю запрос от клиента
            int oper = in.readInt(); // Операция
            int v1   = in.readInt(); // Число 1
            int v2   = in.readInt(); // Число 2
            // Считаю результат
            int result;              // Результат операции
            result = (oper==0)? (v1+v2) : (v1-v2);
            // Отправляю результат клиенту
            out.writeInt(result);
            return true;
        }
        catch(IOException e)
        {
            return false;
        }
    }
```

```
    // главный метод
    public static void main(String[] args)
    {
        try
        {
            Server srv = new Server();
            srv.start(12345);
        }
        catch(IOException e)
        {
            System.out.println("Возникла ошибка");
        }
    }
}
```

Клиентский код:

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

public class Client {
    private Socket sock = null;
    private DataOutputStream out = null;
    private DataInputStream in = null;

    // конструктор
    public Client(String ip, int port) throws IOException
    {
        // Устанавливаем соединение
        sock = new Socket(ip,port);
        // Получаем потоки ввода-вывода
        in = new DataInputStream(sock.getInputStream( ));
        out = new DataOutputStream(sock.getOutputStream());
    }

    // отправить запрос серверу и получить ответ
    private int sendQuery(int operation, int value1,
                int value2) throws IOException
    {
        // отправляю запрос
        out.writeInt(operation);
        out.writeInt(value1);
        out.writeInt(value2);
        // получаю ответ
        int res = in.readInt();
        return res;
    }
```

```
    // посчитать сумму чисел
    public int sum(int value1, int value2) throws IOException
    {
        return sendQuery(0, value1, value2);
    }

    // посчитать разность чисел
    public int sub(int value1, int value2) throws IOException
    {
        return sendQuery(1, value1, value2);
    }

    // отсоединиться
    public void disconnect() throws IOException
    {
        sock.close();
    }

    // главный метод
    public static void main(String[] args) {
        try
        {
            Client client = new Client("localhost",12345);
            int a = client.sum(15,20);
            int b = client.sub(30,38);
            int c = client.sum(100,200);
            System.out.println(a);
            System.out.println(b);
            System.out.println(c);
            client.disconnect();
        }
        catch(IOException e)
        {
            System.out.println("Возникла ошибка");
        }
    }
}
```

2.4.2. Пример 2. Передача данных в формате строки с разделителем

Модифицируем пример 1, чтобы клиент и сервер обменивались сообщениями в формате строки с разделителем и усложним протокол взаимодействия клиента и сервера. Теперь сервер будет проверять, верное ли количество параметров пришло ему на вход, верный ли тип входных параметров, и правильно ли задан код операции.

Протокол взаимодействия будет выглядеть следующим образом:

1. Клиент отправляет серверу запрос в формате строки:
   ”код_операции#операнд1#операнд2”
   В качестве разделителя используется символ ‘#’.

Таблица 1. Структура запроса клиента

Поле	Тип данных	Описание
Код операции	int	Код арифметической операции: 0 – сложение 1 – вычитание
операнд1	int	Первый операнд выражения
операнд1	int	Второй операнд выражения

2. Сервер отправляет клиенту ответ в формате строки, содержащий код завершения и результат вычисления операции:
   `"код_завершения#результат "`

Таблица 2. Структура ответа сервера

Поле	Тип данных	Описание
Код завершения	int	0 – операция завершилась успешно 1 – операция завершилась ошибкой: неверное количество параметров во входной строке 2 – операция завершилась ошибкой: неверный код операции 3 – операция завершилась ошибкой: неверный тип входных параметров
Результат	int	Результат сложения/вычитания двух чисел

В данном примере представлено два приложения клиента.

Клиент 1 (класс Client) – модификация клиента из примера 1, работающая с текстовыми сообщениями.

Клиент 2 (класс SimpleClient) – «клиент-вредитель» – приложение, пытающееся отправить серверу некорректные запросы.

Исходный код приложения-сервера:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {
    private ServerSocket server = null;
    private Socket sock = null;
    private PrintWriter out = null;
    private BufferedReader in = null;

    // запустить сервер
    public void start(int port) throws IOException
    {
        server = new ServerSocket(port);
        while (true)
        {
            // Принимаем соединение от нового клиента
```

```
        sock = server.accept();
        // Получаем потоки ввода-вывода
        in = new BufferedReader(
               new InputStreamReader(sock.getInputStream( )));
        out = new PrintWriter(sock.getOutputStream(), true);
        // Пока соединение активно, обрабатываем запросы
        while (processQuery());
    }
}

// обработка запроса
private boolean processQuery()
{
    int result = 0; // Результат операции
    int comp_code = 0; // Код завершения
    try
    {
        // Получаю запрос от клиента
        String query = in.readLine();
        if (query==null) return false;
        // Разбиваю строку на поля, разделенные через $
        String[] fields = query.split("#");
        if (fields.length != 3)
        {
            comp_code = 1; // неверное количество параметров
        }
        else
        {
            try
            {
                int oper = Integer.valueOf(fields[0]);
                int v1   = Integer.valueOf(fields[1]);
                int v2   = Integer.valueOf(fields[2]);
                if (oper == 0)
                    result = v1+v2; // операция - сложение
                else if (oper == 1)
                    result = v1-v2; // операция - вычитание
                else
                    comp_code = 2; // неверный код операции

            } catch (NumberFormatException e)
            {
                comp_code = 3; // неверный тип параметров
            }
        }
        // Формирую ответ
        String response = comp_code+"#"+result;
        // Отправвляю клиенту
        out.println(response);
        return true;
    }
    catch(IOException e)
    {
```

```
            return false;
        }
    }

    // главный метод
    public static void main(String[] args)
    {
        try
        {
            Server srv = new Server();
            srv.start(12345);
        }
        catch(IOException e)
        {
            System.out.println("Возникла ошибка");
        }
    }
}
```

Клиент 1 (исходный код):

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class Client {
    private Socket sock = null;
    private PrintWriter out = null;
    private BufferedReader in = null;

    // конструктор
    public Client(String ip, int port) throws IOException
    {
        // Устанавливаем соединение
        sock = new Socket(ip,port);
        // Получаем потоки ввода-вывода
        in = new BufferedReader(
             new InputStreamReader(sock.getInputStream( )));
        out = new PrintWriter(sock.getOutputStream(), true);
    }

    // отправить запрос серверу и получить ответ
    private int sendQuery(int operation, int value1,
                int value2) throws IOException
    {
        // формирую запрос
        String query = operation+"#"+value1+"#"+value2;
        // отправляю запрос
        out.println(query);
        // получаю ответ
```

```
        String response = in.readLine();
        String[] fields = response.split("#");
        if (fields.length!=2)
            throw new IOException("Invalid response from server");
        try
        {
            // Код завершения
            int comp_code = Integer.valueOf(fields[0]);
            // Результат операции
            int result = Integer.valueOf(fields[1]);
            if (comp_code==0)
                return result;
            else
                throw new IOException(
                    "Error while processing query");
        }
        catch(NumberFormatException e)
        {
            throw new IOException("Invalid response from server");
        }
    }

    // посчитать сумму чисел
    public int sum(int value1, int value2) throws IOException
    {
        return sendQuery(0, value1, value2);
    }

    // посчитать разность чисел
    public int sub(int value1, int value2) throws IOException
    {
        return sendQuery(1, value1, value2);
    }

    // отсоединиться
    public void disconnect() throws IOException
    {
        sock.close();
    }

    // главный метод
    public static void main(String[] args) {
        try
        {
            Client client = new Client("localhost",12345);
            int a = client.sum(15,20);
            int b = client.sub(30,38);
            int c = client.sum(10,20);
            System.out.println(a);
            System.out.println(b);
            System.out.println(c);
            client.disconnect();
        }
```

```
        catch(IOException e)
        {
            System.out.println("Возникла ошибка");
            e.printStackTrace();
        }
    }
}
```

Клиент 2 (исходный код):

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class SimpleClient {
    private Socket sock = null;
    private PrintWriter out = null;
    private BufferedReader in = null;

    public void test(String ip, int port) throws IOException
    {
        // Устанавливаем соединение
        sock = new Socket(ip,port);
        // Получаем потоки ввода-вывода
        in = new BufferedReader(
             new InputStreamReader(sock.getInputStream( )));
        out = new PrintWriter(sock.getOutputStream(), true);

        // Отправляю тестовые запросы, получаю ответы,
        // вывожу их на экран
        out.println("бубубу");
        System.out.println(in.readLine());

        out.println("xxx#yyy#zzz");
        System.out.println(in.readLine());

        out.println("10#20#30");
        System.out.println(in.readLine());

        out.println("0#5#7");
        System.out.println(in.readLine());

        out.println("1#5#7");
        System.out.println(in.readLine());
    }

    public static void main(String[] args) throws IOException
    {
        (new SimpleClient()).test("localhost",12345);
    }
}
```

Клиент 2 (результат работы):

```
1#0
3#0
2#0
0#12
0#-2
```