Глава 13

СЕТЕВЫЕ ПРОГРАММЫ

Если неправильно набрать номер,
никогда не будет гудков «занято».

Загадка Ковака

Поддержка Интернета

Язык Java делает сетевое программирование простым благодаря наличию специальных средств и классов. Большинство этих классов находится в пакете **java.net**. Сетевые классы имеют методы для установки сетевых соединений передачи запросов и сообщений. Многопоточность позволяет обрабатывать несколько соединений. Сетевые приложения используют интернет-приложения, к которым относятся веб-браузер, e-mail, сетевые новости, передача файлов. Для создания таких приложений используются сокеты, порты, протоколы TCP/IP, UDP.

Приложения клиент/сервер используют компьютер, выполняющий специальную программу-сервер, которая обычно устанавливается на удаленном компьютере и предоставляет услуги другим программам-клиентам. Клиент — это программа, получающая услуги от сервера. Клиент устанавливает соединение с сервером и пересылает серверу запрос. Сервер осуществляет прослушивание клиентов, получает и выполняет запрос после установки соединения. Результат выполнения запроса может быть возвращен сервером клиенту. Запросы и сообщения представляют собой записи, структура которых определяется используемыми протоколами.

В стеке протоколов TCP/IP используются следующие прикладные протоколы:

HTTP(s) — Hypertext Transfer Protocol (WWW);
NNTP — Network News Transfer Protocol (группы новостей);
SMTP — Simple Mail Transfer Protocol (посылка почты);
POP3 — Post Office Protocol (чтение почты с сервера);
FTP — File Transfer Protocol (протокол передачи файлов).

Каждый компьютер из подключенных к сети по протоколу TCP/IP имеет уникальный IP-адрес, используемый для идентификации и установки соединения. IP-адрес существует в двух видах: IPv4 и IPv6. Формат IPv4 представлен 32-битовым числом, обычно записываемым как четыре числа, разделенные

точками, каждое из которых изменяется от **0** до **255**, например **217.21.43.10**. Формат IPv6 представлен 128-битовым числом, обычно записываемым как восемь шестнадцатеричных чисел, разделенных двоеточиями, например **1170:0:0:0:7:771:100A:214B**. IP-адрес может быть временным и выделяться динамически для каждого подключения или быть постоянным, как для сервера. IP-адреса используются во внутренних сетевых системах. Обычно при подключении к Интернету вместо числового IP-адреса используются символьные имена (например: www.bsu.by), называемые именами домена. Специальная программа DNS (Domain Name Server), располагаемая на отдельном сервере, проверяет адрес и преобразует имя домена в числовой IP-адрес. Если в качестве сервера используется этот же компьютер без сетевого подключения, в качестве IP-адреса указывается **127.0.0.1** или **localhost**. Для явной идентификации услуг к IP-адресу присоединяется номер порта через двоеточие, например **217.21.43.10:443**. Здесь указан номер порта **443**. Номера портов от **1** до **1024** могут быть заняты для внутреннего использования, например, если порт явно не указан, браузер воспользуется значением по умолчанию: **20** — **FTP**-данные, **21** — **FTP**-управление, **53** — **DNS**, **80** — **HTTP**, **25** — **SMTP**, **110** — **POP3**, **119** — **NNTP**. К серверу можно подключиться с помощью различных портов. Каждый порт указывает конкретное место соединения на указанном компьютере и предоставляет определенную услугу.

Для доступа к интернет-ресурсу в браузере указывается адрес URL. Адрес URL (Universal Resource Locator) состоит из двух частей — префикса протокола (http, https, ftp и т. д.) и URI (Universal Resource Identifier). URI содержит интернет-адрес, необязательный номер порта и путь к каталогу, содержащему файл, например:

```
http://www.oracle.com/download.jsp
```

URI не может содержать такие специальные символы, как пробелы, табуляции, возврат каретки. Их можно задавать через шестнадцатеричные коды. Например: **%20** обозначает пробел. Другие зарезервированные символы: символ **&** — разделитель аргументов, символ **?** следует перед аргументами запросов, символ **+** — пробел, символ **#** — ссылки внутри страницы, например *имя_страницы#имя_ссылки*.

Определить IP-адрес в приложеннии можно с помощью объекта класса **java.net.InetAddress**. Можно также использовать и специфические классы **Inet4Address** и **Inet6Address**.

Класс **InetAddress** не имеет **public**-конструкторов. Создать объект класса можно с помощью статических методов. Метод **getLocalHost()** возвращает объект класса **InetAddress**, содержащий IP-адрес и имя компьютера, на котором выполняется программа. Метод **getByName(String host)** возвращает объект класса **InetAddress**, содержащий IP-адрес по имени компьютера, используя пространство имен DNS. IP-адрес может быть временным, различным для

каждого соединения, однако он остается постоянным, если соединение установлено. Метод **getByAddress(byte[] addr)** создает объект класса **InetAddress**, содержащий имя компьютера, по IP-адресу, представленному в виде массива байт. Если компьютер имеет несколько IP, то получить их можно методом **getAllByName(String host)**, возвращающим массив объектов класса **InetAddress**. Если IP для данной машины один, то массив будет содержать один элемент. Метод **getByAddress(String host, byte[] addr)** создает объект класса **InetAddress** с заданным именем и IP-адресом, не проверяя существование такого компьютера. Все эти методы являются потенциальными генераторами исключительной ситуации **UnknownHostException**, и поэтому их вызов должен быть обработан с помощью **throws** для метода или блока **try-catch**. Проверить доступ к компьютеру в данный момент можно с помощью метода **boolean isReachable(int timeout)**, который возвращает **true**, если компьютер доступен, где **timeout** — время ожидания ответа от компьютера в миллисекундах.

Следующая программа демонстрирует при наличии Internet-соединения, как получить IP-адрес текущего компьютера и IP-адрес из имени домена с помощью сервера имен доменов (DNS), к которому обращается метод **getByName()** класса **InetAddress**.

/ # 1 # вывод IP-адреса компьютера и Интернет-ресурса # InetLogic.java*/*

```
package by.bsu.net;
import java.net.InetAddress;
public class InetLogic {
          public static void main(String[] args) {
                    InetAddress currentIP = null;
                    InetAddress bsuIP = null;
                    try {
                              currentIP = InetAddress.getLocalHost();
                              // вывод IP-адреса локального компьютера
                              System.out.println("Мой IP -> " + currentIP.getHostAddress());
                              bsuIP = InetAddress.getByName("www.bsu.by");
                              // вывод IP-адреса БГУ www.bsu.by
                              System.out.println("BSU -> " + bsuIP.getHostAddress());
                    } catch (UnknownHostException e) {
                              // если не удается найти IP
                              System.err.println("адрес недоступен " + e);
                    }
          }
}
```

В результате будет выведено, например:

Мой IP -> 172.17.16.14
BSU -> 217.21.43.10

Метод **getLocalHost()** класса **InetAddress** создает объект **currentIP** и возвращает IP-адрес компьютера.

/ # 2 # присваивание фиктивного имени реальному ресурсу, заданному через IP # UnCheckedHost.java */*

```
package by.bsu.net;
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;
public class UnCheckedHost {
    public static void main(String[] args) {
              // задание IP-адреса в виде массива
              byte ip[] ={(byte)217, (byte)21, (byte)43, (byte)10};
              try {
                        InetAddress addr = InetAddress.getByAddress("University", ip);
                        System.out.println(addr.getHostName()
                          + "-> соединение:" + addr.isReachable(1000));
              } catch (UnknownHostException e) {
                        System.err.println("адрес недоступен " + e);
              } catch (IOException e) {
                        System.err.println("ошибка потока " + e);
              }
    }
}
```

В результате будет выведено в случае подключения к сети Интернет:

University-> соединение:true

Для доступа к ресурсам можно создать объект класса **URL**, указывающий на ресурсы в Интернете. В следующем примере объект **URL** используется для доступа к HTML-файлу, на который он указывает и отображает его в окне браузера с помощью метода **showDocument()**.

/ # 3 # запуск страницы из апплета # ShowDocument.java */*

```
package by.bsu.net;
import java.awt.Graphics;
import java.net.MalformedURLException;
import java.net.URL;
import javax.swing.JApplet;
public class ShowDocument extends JApplet {
          private URL bsu = null;
          public String getBaseURL() {
                    return "http://www.bsu.by";
          }
          public void paint(Graphics g) {
                    int timer = 0;
                    g.drawString("Загрузка страницы", 10, 10);
                    try {
                              for (; timer < 30; timer++) {
                                   g.drawString(".", 10 + timer * 5, 25);
```

```
                                Thread.sleep(100);
                            }
                            bsu = new URL(getBaseURL());
                            this.getAppletContext().showDocument(bsu, "_blank");
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        } catch (MalformedURLException e) {
                        // некорректно задан протокол, доменное имя или путь к файлу
                            e.printStackTrace();
                        }
                }
}
```

Метод **showDocument()** может содержать параметры для отображения страницы различными способами: «**_self**» — выводит документ в текущий фрейм, «**_blank**» — в новое окно, «**_top**» — на все окно, «**_parent**» — в родительском окне, «**имя_окна**» — в окне с указанным именем. Для корректной работы данного примера апплет следует запускать из браузера, используя следующий HTML-документ:

```
<html>
<body align=center>
<applet code= by.bsu.net.ShowDocument.class></applet>
</body></html>
```

В следующей программе читается содержимое HTML-файла по указанному адресу и выводится в окно консоли.

/ # 4 # чтение документа из Интернета # ReadDocument.java */*

```
package by.bsu.net;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
public class ReadDocument {
          public static void main(String[] args) {
                    URL bsu = null;
                    String urlName = "http://www.bsu.by";
                    try {
                              bsu = new URL(urlName);
                    } catch (MalformedURLException e) {
                    // некорректно заданы протокол, доменное имя или путь к файлу
                              e.printStackTrace();
                    }
                    if (bsu == null) {
                              throw new RuntimeException();
                    }
                    try (BufferedReader d = new BufferedReader(new InputStreamReader(
```

```
                                        bsu.openStream()))) {
                                String line = "";
                                while ((line = d.readLine()) != null) {
                                        System.out.println(line);
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
}
```

Для получения более полной информации о ресурсе требуется применять класс **URLConnection**. Для получения экземпляра следует вызвать на экземпляре класса **URL** метод **openConnection()**. Далее при необходимости можно указать значения некоторых свойств. Для установления соединения на полученном экземпляре **URLConnection** вызывается метод **connect()**. При вызове этого метода возможна серьезная блокировка основного потока, поэтому перед попыткой установления соединения следует установить таймаут на соединение и/или на чтение методами **setConnectTimeout(int timeout)** и **setReadTimeout(int timeout)**.

/ # 5 # инфомация об Интернет-ресурсе # ConnectionDemo.java */*

```
package by.bsu.net;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
public class ConnectionDemo {
    public static void main(String[] args) {
            String urlName = "http://www.oracle.com";
            int timeout = 10_000_000;
            URL url;
            try {
                    url = new URL(urlName);
                    final URLConnection connection = url.openConnection();
                    // установка таймаута на соединение
                    connection.setConnectTimeout(timeout);
                    System.out.println(urlName + " content type: "
                                                    + connection.getContentType());
                    // продолжение извлечения информации из соединения
            } catch (MalformedURLException e) {
                    e.printStackTrace();
            } catch (IOException e) {
                    e.printStackTrace();
            }
    }
}
```

В результате при успешной установке соединения будет выведено:

http://www.oracle.com content type: text/html;charset=ISO-8859-1

Сокетные соединения по протоколу TCP/IP

Сокеты (сетевые разъемы) — это логическое понятие, соответствующее разъемам, к которым подключены сетевые компьютеры и через которые осуществляется двунаправленная поточная передача данных между компьютерами. Сокет определяется номером порта и IP-адресом. При этом IP-адрес используется для идентификации компьютера, номер порта — для идентификации процесса, работающего на компьютере. Когда одно приложение знает сокет другого, создается сокетное протоколо-ориентированное соединение по протоколу TCP/IP. Клиент пытается соединиться с сервером, инициализируя сокетное соединение. Сервер прослушивает сообщение и ждет, пока клиент не свяжется с ним. Первое сообщение, посылаемое клиентом на сервер, содержит сокет клиента. Сервер, в свою очередь, создает сокет, который будет использоваться для связи с клиентом, и посылает его клиенту с первым сообщением. После этого устанавливается коммуникационное соединение.

Сокетное соединение с сервером создается клиентом с помощью объекта класса **Socket**. При этом указывается IP-адрес сервера и номер порта. Если указано символьное имя домена, то Java преобразует его с помощью DNS-сервера к IP-адресу. Например, если сервер установлен на этом же компьютере, соединение с сервером можно установить из приложения клиента с помощью инструкции:

```
Socket socketClient = new Socket("ИМЯ_СЕРВЕРА", 8030);
```

Сервер ожидает сообщения клиента и должен быть заранее запущен с указанием определенного порта. Объект класса **ServerSocket** создается с указанием конструктору номера порта и ожидает сообщения клиента с помощью метода **accept()** класса **ServerSocket**, который возвращает сокет клиента:

```
ServerSocket server = new ServerSocket(8030);
Socket socketServ = server.accept();
```

Таким образом, для установки необходимо установить IP-адрес и номер порта сервера, IP-адрес и номер порта клиента. Обычно порт клиента и сервера устанавливаются одинаковыми. Клиент и сервер после установления сокетного соединения могут получать данные из потока ввода и записывать данные в поток вывода с помощью методов **getInputStream()** и **getOutputStream()** или к **PrintStream** для того, чтобы программа могла трактовать поток как выходные файлы.

В следующем примере для отправки клиенту строки «*привет!*» сервер вызывает метод **getOutputStream()** класса **Socket**. Клиент получает данные от сервера

с помощью метода **getInputStream()**. Для разъединения клиента и сервера после завершения работы сокет закрывается с помощью метода **close()** класса **Socket**. В данном примере сервер отправляет клиенту строку, после чего разрывает связь.

/ # 6 # передача клиенту строки # SmallServerSocket.java */*

```
package by.bsu.net;
import java.io.IOException;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket
public class SmallServerSocket {
          public static void main(String[] args) {
                    Socket s = null;
                    PrintStream ps = null;
                    try { // отправка строки клиенту
                              // создание объекта и назначение номера порта
                              ServerSocket server = new ServerSocket(8030);
                              s = server.accept(); // ожидание соединения
                              ps = new PrintStream(s.getOutputStream());
                              // помещение строки "привет!" в буфер
                              ps.println("привет!");
                              // отправка содержимого буфера клиенту
                              ps.flush();
                    } catch (IOException e) {
                              System.err.println("Ошибка соединения потока: " + e);
                    } finally {
                        if (ps != null) {
                              ps.close();
                        }
                        if (s != null) {
                              try { // разрыв соединения
                                        s.close();
                              } catch (IOException e) {
                                        e.printStackTrace();
                              }
                        }
                    }
          }
}
```

/ # 7 # получение клиентом строки # SmallClientSocket.java */*

```
package by.bsu.net;
import java.io.*;
import java.net.*;
public class SmallClientSocket {
          public static void main(String[] args) {
                    Socket socket = null;
```

```
        try { // получение строки клиентом
                socket = new Socket("ИМЯ_СЕРВЕРА", 8030);
                                /* здесь "ИМЯ_СЕРВЕРА" компьютер,
                                на котором стоит сервер-сокет"*/
                BufferedReader br = new BufferedReader(new InputStreamReader(
                                                socket.getInputStream()));
                String message = br.readLine();
                        System.out.println(message);
                } catch (IOException e) {
                        System.err.println("ошибка: " + e);
                } finally {
                    if (br != null) {
                        br.close();
                    }
                        if (socket != null) {
                        try { // разрыв соединения
                                socket.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                       }
                }
        }
}
```

Аналогично клиент может послать данные серверу через поток вывода, полученный с помощью метода **getOutputStream()**, а сервер может получать данные через поток ввода, полученный с помощью метода **getInputStream()**.

Если необходимо протестировать подобный пример на одном компьютере, можно выступать одновременно в роли клиента и сервера, используя статические методы **getLocalHost()** класса **InetAddress** для получения динамического IP-адреса компьютера, который выделяется при входе в сеть Интернет.

Многопоточность

Сервер должен поддерживать многопоточность, иначе он будет не в состоянии обрабатывать несколько соединений одновременно. В этом случае сервер содержит цикл, ожидающий нового клиентского соединения. Каждый раз, когда клиент просит соединения, сервер создает новый поток. В следующем примере создается класс **ServerThread**, расширяющий класс **Thread**, и используется затем для соединений с многими клиентами, каждый в своем потоке.

/ # 8 # сервер для множества клиентов # NetServerThread.java */*

```
package by.bsu.net;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.io.PrintStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
public class NetServerThread {
    public static void main(String[] args) {
          try {
                    ServerSocket server = new ServerSocket(8071);
                    System.out.println("initialized");
                    while (true) {
                              // ожидание клиента
                              Socket  socket  =  server.accept();
                              System.out.println(socket.getInetAddress().getHostName() + " connected");
                              /*
                                 создание отдельного потока для обмена данными
                                 с соединившимся клиентом
                               */
                              ServerThread thread = new ServerThread(socket);
                              // запуск потока
                              thread.start();
                    }
          } catch (IOException e) {
                    System.err.println(e);
          }
    }
}

class ServerThread extends Thread {
    private PrintStream os; // передача
    private BufferedReader is; // прием
    private InetAddress addr; // адрес клиента
    public ServerThread(Socket s) throws IOException {
              os = new PrintStream(s.getOutputStream());
              is = new BufferedReader(new InputStreamReader(s.getInputStream()));
              addr = s.getInetAddress();
    }
    public void run() {
              int i = 0;
              String str;
              try {
                  while ((str = is.readLine()) != null) {
                            if ("PING".equals(str)) {
                                      os.println("PONG " + ++i);
                            }
                            System.out.println("PING-PONG " + i + " with " + addr.getHostName());
              }
              }catch (IOException e) {
                        // если клиент не отвечает, соединение с ним разрывается
                        System.err.println("Disconnect");
```

```
                } finally {
                        disconnect(); // уничтожение потока
                }
        }
        public void disconnect() {
                try {
                        if (os != null) {
                                os.close();
                        }
                        if (is != null) {
                                is.close();
                        }
                        System.out.println(addr.getHostName() + " disconnecting");
                } catch (IOException e) {
                        e.printStackTrace();
                } finally {
                        this.interrupt();
                }
        }
}
```

Сервер передает сообщение клиенту. Для клиентских приложений поддержка многопоточности также необходима. Например, один поток ожидает выполнения операции ввода/вывода, а другие потоки выполняют свои функции.

/ # 9 # получение и передача сообщения клиентом в потоке # NetClient.java */*

```
package by.bsu.net;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
public class NetClient {
    public static void main(String[] args) {
        Socket socket = null;
        BufferedReader br = null;
        try {
            // установка соединения с сервером
            socket = new Socket(InetAddress.getLocalHost(), 8071);
            // или Socket socket = new Socket("ИМЯ_СЕРВЕРА", 8071);
            PrintStream ps = new PrintStream(socket.getOutputStream());
            br = new BufferedReader(new InputStreamReader( socket.getInputStream()));
            for (int i = 1; i <= 10; i++) {
                ps.println("PING");
                System.out.println(br.readLine());
                Thread.sleep(1_000);
```

```
                    }
                } catch (UnknownHostException e) {
                    // если не удалось соединиться с сервером
                    System.err.println("адрес недоступен" + e);
                } catch (IOException e) {
                    System.err.println("ошибка I/O потока" + e);
                } catch (InterruptedException e) {
                    System.err.println("ошибка потока выполнения" + e);
                } finally {
                    if (br != null) {
                        try {
                            br.close();
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                    if (socket!= null) {
                        try {
                            socket.close();
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                }
            }
}
```

Сервер должен быть инициализирован до того, как клиент попытается осуществить сокетное соединение. При этом может быть использован IP-адрес локального компьютера.

Датаграммы и протокол UDP

Протокол UDP (User Datagram Protocol) не устанавливает виртуального соединения и не гарантирует доставки данных. Отправитель просто посылает пакеты по указанному адресу; если отосланная информация была повреждена или вообще не дошла, отправитель об этом даже не узнает. Однако достоинством UDP является высокая скорость передачи данных. Данный протокол часто используется при трансляции аудио- и видеосигналов, где потеря небольшого количества данных не может привести к серьезным искажениям всей информации.

По протоколу UDP данные передаются пакетами. Пакетом в этом случае UDP является объект класса **DatagramPacket**. Этот класс содержит в себе передаваемые данные, представленные в виде массива байт.

Конструктор **DatagramPacket(byte[] buf, int length)** используется в тех случаях, когда датаграмма только принимает в себя пришедшие данные, так как

созданный с его помощью объект не имеет информации об адресе и порте получателя. Конструктор **DatagramPacket(byte[] buf, int length, InetAddress address, int port)** используется для отправки датаграмм.

Класс **DatagramSocket** может выступать в роли клиента и сервера, т. е. он способен получать и отправлять пакеты. Отправить пакет можно с помощью метода **send(DatagramPacket pac)**, для получения пакета используется метод **receive(DatagramPacket pac)**.

/ # 10 # передача файла по протоколу UDP # UDPSender.java */*

```
package by.bsu.net;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;
public class UDPSender {
          public static void main(String[] args) {
                    String fileName = "audio/toxic.mp3";
                    try (FileInputStream fr = new FileInputStream(new File(fileName))) {
                              byte[] data = new byte[1024];
                              DatagramSocket dSocket = new DatagramSocket();
                              InetAddress address = InetAddress.getLocalHost();
                              DatagramPacket packet;
                              while (fr.read(data) != -1) {
                                        // создание пакета данных
                                  packet = new DatagramPacket(data, data.length, address, 8033);
                                  dSocket.send(packet);// передача пакета
                              }
                              System.out.println("Файл успешно отправлен");
                    } catch (UnknownHostException e) {
                              // неверный адрес получателя
                              e.printStackTrace();
                    } catch (SocketException e) {
                              // возникли ошибки при передаче данных
                              e.printStackTrace();
                    } catch (FileNotFoundException e) {
                              e.printStackTrace();
                    } catch (IOException e) {
                              e.printStackTrace();
                    }
          }
}
```

/ # 11 # прием данных по протоколу UDP # UDPRecipient.java */*

```
package by.bsu.net;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketTimeoutException;
public class UDPRecipient {
          public static void main(String[] args) {
                    File file = new File("UDPtoxic.mp3");
                    System.out.println("Прием данных...");
                    // прием файла
                    acceptFile(file, 8033, 1024);
          }
          private static void acceptFile(File file, int port, int pacSize)  {
                    byte data[] = new byte[pacSize];
                    DatagramPacket pac = new DatagramPacket(data, data.length);
                    try (FileOutputStream os = new FileOutputStream(file)) {
                              DatagramSocket s = new DatagramSocket(port);
                              /* установка времени ожидания: если в течение 60 секунд
                              не принято ни одного пакета, прием данных заканчивается */
                              s.setSoTimeout(60_000);
                              while (true) {
                                        s.receive(pac);
                                        os.write(data);
                                        os.flush();
                              }
                    } catch (SocketTimeoutException e) {
                              // если время ожидания вышло
                       System.err.println("Истекло время ожидания, прием данных закончен");
                    } catch (FileNotFoundException e) {
                              e.printStackTrace();
                    } catch (IOException e) {
                              e.printStackTrace();
                    }
          }
}
```

Задания к главе 13

Вариант А

Создать на основе сокетов клиент/серверное визуальное приложение:

1. Клиент посылает через сервер сообщение другому клиенту.
2. Клиент посылает через сервер сообщение другому клиенту, выбранному из списка.

3. Чат. Клиент посылает через сервер сообщение, которое получают все клиенты. Список клиентов хранится на сервере в файле.
4. Клиент при обращении к серверу получает случайно выбранный сонет Шекспира из файла.
5. Сервер рассылает сообщения выбранным из списка клиентам. Список хранится в файле.
6. Сервер рассылает сообщения в определенное время определенным клиентам.
7. Сервер рассылает сообщения только тем клиентам, которые в настоящий момент находятся в on-line.
8. Чат. Сервер рассылает всем клиентам информацию о клиентах, вошедших в чат и покинувших его.
9. Клиент выбирает изображение из списка и пересылает его другому клиенту через сервер.
10. Игра по сети в «Морской бой».
11. Игра по сети в «21».
12. Игра по сети «Го». Крестики-нолики на безразмерном (большом) поле. Для победы необходимо выстроить пять в один ряд.
13. Написать программу, сканирующую сеть в указанном диапазоне IP адресов на наличие активных компьютеров.
14. Прокси. Программа должна принимать сообщения от любого клиента, работающего на протоколе TCP, и отсылать их на соответствующий сервер. При передаче изменять сообщение.
15. Телнет. Создать программу, которая соединяется с указанным сервером по указанному порту и производит обмен текстовой информацией.

Вариант В

Для заданий варианта В гл. 4 на базе сокетных соединений разработать сетевой вариант системы. Для каждого пользователя должно быть создано клиентское приложение, соединяющееся с сервером.