**Лабораторная работа №4**

**Программирование сокетов**

**Клиент-сервер (англ.Client-server)** — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемыми серверами, и заказчиками услуг, называемыми клиентами. Нередко клиенты и серверы взаимодействуют через компьютерную сеть и могут быть как различными физическими устройствами, так и программным обеспечением. Таким образом, работа сервера состоит в прослушивании соединения, она выполняется с помощью специального объекта, который вы создаете. Работа клиента состоит в попытке создать соединение с сервером, и это выполняется с помощью специального клиентского объекта, который вы создаете. Как только соединение установлено, вы увидите, что и клиентская, и серверная сторона соединения превращаются в потоковый объект ввода/вывода, таким образом вы можете трактовать соединение, как будто вы читаете и пишете файл. Таким образом, после установки соединения, вы просто используете хорошо знакомые команды ввода/вывода. Это одна из особенностей работы по сети в Java.

**Сокет** - это программный интерфейс, предназначенный для передачи данных между приложениями. Что же касается типов сокетов, то их два - потоковые и датаграммные. В Java вы создаете сокет, чтобы создать соединение с другой машиной, затем вы получаете InputStream и OutputStream (или, с соответствующими конверторами, Reader и Writer) из сокета, чтобы получить возможность трактовать соединение, как объект потока ввода/вывода.

Существует два класса сокетов, основанных на потоках: ServerSocket, который использует сервер для "прослушивания" входящих соединений, и Socket, который использует клиент для инициализации соединения. Как только клиент создаст сокетное соединение, ServerSocket возвратит (посредством метода accept( )) соответствующий Socket, через который может происходить коммуникация на стороне сервера. На этой стадии вы используете методы getInputStream( ) и getOutputStream( ) для получения соответствующих объектов InputStream'а и outputStream'а для каждого сокета. Они должны быть обернуты внутрь буферных и форматирующих классов точно так же, как и другие объекты потоков.

Когда вы создаете ServerSocket, вы даете ему только номер порта. Вы не даете ему IP адрес, поскольку он уже есть на той машине, на которой он установлен. Однако когда вы создаете Socket, вы должны передать ему и IP адрес, и номер порта, с которым вы хотите соединиться. Socket, который возвращается из метода ServerSocket.accept( ) уже содержит всю эту информацию.

Пример создания простейшего сервера, принимающего от клиента сообщения. Окончание сеанса – слово END:

import java.io.\*;  
import java.net.\*;  
public class ExampleServer {  
  
 public static final int *PORT* = 8088;  
 public static void main(String[] args) throws  
 IOException {  
 ServerSocket s = new ServerSocket(*PORT*);  
 System.*out*.println("Started: " + s);  
 try {  
 Socket socket = s.accept();  
 try {  
 System.*out*.println("Connection accepted: " + socket);  
 BufferedReader in = new BufferedReader(new InputStreamReader(  
 socket.getInputStream()));  
 PrintWriter out = new PrintWriter(new  
 BufferedWriter(  
 new OutputStreamWriter(socket.getOutputStream())), true);  
 while (true) {  
 String str = in.readLine();  
 if (str.equals("END"))  
 break;  
 System.*out*.println("Receiving: " + str);  
 out.println(str);  
 }  
  
 }  
 finally {  
 System.*out*.println("closing...");  
 socket.close();  
 }  
 }  
 finally {  
 s.close();  
 }  
 }  
}

Клиентская часть создается аналогичным образом.

**Задание.**

Создать на основе сокетов клиент/серверное приложение «Чат». Клиент регистрируется на сервере, после чего посылает через сервер сообщения, которые получают все клиенты. Список клиентов хранится на сервере.

Интерфейс у клиентского приложения определить самостоятельно.