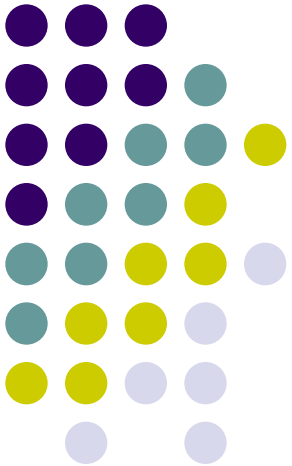
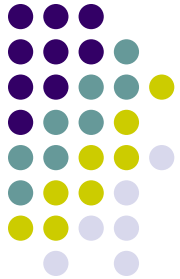
UDP сокеты



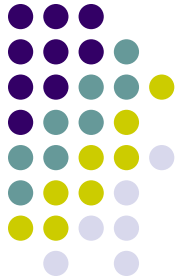
Протокол пользовательских дейтаграмм(UDP)



- Протокол UDP намного проще, чем TCP; он полезен в ситуациях, когда мощные механизмы обеспечения надежности протокола TCP не обязательны. Заголовок UDP имеет всего четыре поля:
  - поле порта источника (*source port*),
  - поле порта пункта назначения (*destination port*),
  - поле длины (*length*) и
  - поле контрольной суммы UDP (*checksum UDP*).

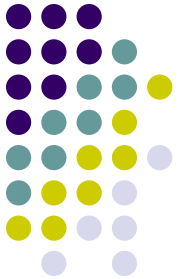
  Поля порта источника и порта назначения выполняют те же функции, что и в заголовке TCP. Поле длины обозначает длину заголовка UDP и данных; поле контрольной суммы обеспечивает проверку целостности пакета. Контрольная сумма UDP является факультативной возможностью.

Дейтаграмма



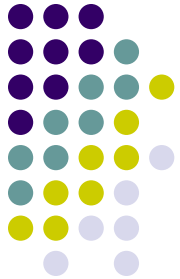
- ***Дейтаграмма*** - это *пакет*, передаваемый через сеть независимо от других *пакетов* без установления логического соединения и подтверждения приема. ***Дейтаграмма*** - совершенно самостоятельный *пакет*, поскольку сама содержит всю необходимую для ее передачи информацию. Ее передача происходит безо всякого предварения и подготовки.

Дейтаграмма



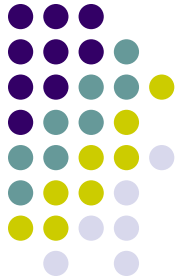
- *Дейтаграммы*, сами по себе, не содержат средств обнаружения и исправления ошибок передачи, поэтому при передаче данных с их помощью следует принимать меры по обеспечению надежности пересылки информации. Методы организации надежности могут быть самыми разными, обычно же используется метод подтверждения приема посылкой эхо-отклика при получении каждого *пакета* с *дейтаграммой*.

Использование UDP



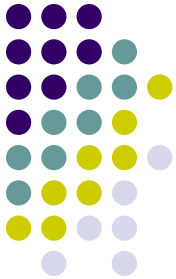
- В отличие от *TCP*, данные, отправляемые прикладным процессом через модуль *UDP*, достигают места назначения как единое целое. Размер каждого записанного сообщения будет совпадать с размером соответствующего прочитанного. *Протокол* UDP сохраняет границы сообщений, определяемые прикладным процессом. Он никогда не объединяет несколько сообщений в одно целое и не делит одно сообщение на части.

Использование UDP и TCP



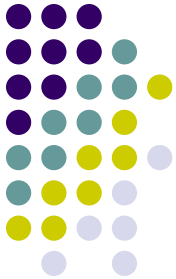
- Альтернатива TCP-UDP позволяет программисту гибко и рационально использовать предоставленные *ресурсы*, исходя из своих возможностей и потребностей.
- Если нужна надежная доставка, то лучше использовать TCP.
- Если нужна эффективность на быстрых сетях с короткими соединениями, то - UDP.
- Прикладные программы, конечно, могут устранять некоторые недостатки выбранного *протокола*. Например, если вы выбрали UDP, а вам необходима надежность, то прикладная программа должна обеспечить надежность сама, как описано выше: требовать подтверждения, пересылки утерянных или увечных *пакетов* и т.д. Если вы выбрали TCP, а вам нужно передавать записи, то прикладная программа должна вставлять метки в поток байтов так, чтобы можно было различить записи.

Свойства протокола UDP



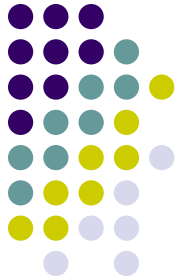
- **Ненадежный** – UDP не имеет ни встроенного механизма обнаружения ошибок, ни средств повторной пересылки поврежденных или потерянных данных
- **Без установления логического соединения** – перед пересылкой данных UDP не устанавливает логического соединения. Информация пересылается в предположении, что принимающая сторона ее ожидает.
- **Основанный на сообщениях** – UDP позволяет приложениям пересылать информацию в виде сообщений, передаваемых посредством дейтаграмм, которые являются единицами передачи данных в UDP. Приложение должно самостоятельно распределить данные по отдельным дейтаграммам.

UDP-сокеты в Java



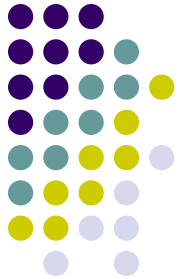
- Java обеспечивает работу с дейтаграммами благодаря использованию классов **DatagramSocket** и **DatagramPacket** пакета java.net.
- С помощью объекта **DatagramSocket** в Java-программе создается сокет для обеспечения сетевого соединения и пересылки дейтаграмм.
- С другой стороны, сама дейтаграмма представляет собой объект класса **DatagramPacket** и используется группой методов класса DatagramSocket.

Создание соединения



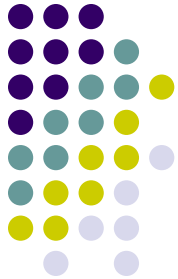
- Для того, чтобы создать новое соединение для пересылки дейтаграмм, следует объявить экземпляр класса DatagramSocket, используя один из следующих его конструкторов:
  - **DatagramSocket();**
  - **DatagramSocket(int port);**
  - **DatagramSocket(int port, InetAddress laddr);**
- Где port – номер порта, laddr-адрес подключения.

Методы класса DatagramSocket



- Для экземпляров класса **DatagramSocket** определена группа методов, позволяющих как управлять процессами сетевого соединения, так и осуществлять его информационную поддержку.
  - **getInetAddress()** – возвращает адрес, к которому осуществляется подключение;
  - **getPort()** – возвращает порт, к которому осуществляется подключение;
  - **getLocalAddress()** – возвращает локальный адрес компьютера, с которого выполняется подключение;
  - **getLocalPOrt()** – возвращает локальный порт, через который производится подключение.

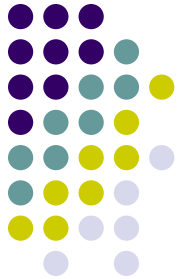
Отправка и получение пакетов DatagramPacket.



- После установки соединения, используя методы **send()** и **receive()**, можно выполнить соответственно отправку и получение пакетов **DatagramPacket.**
- Однако предварительно необходимо будет объявить экземпляры класса **DatagramPacket** виду того, что методы **send()** и **receive()** в качестве параметра принимают объект данного класса. В связи с этим для создания нового экземпляра класса DatagramPacket необходимо будет воспользоваться следующим из его конструкторов:
  - **DatagramPacket(byte[] buf, int length, InetAddress, int port),**

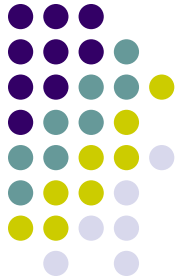
где buf – массив байт, представляющий пакет дейтаграммы, length – размер данного пакета, address-Internet-адрес отправки, port- порт компьютера, на который отправляется дейтаграмма.

Содержимое дейтаграммы



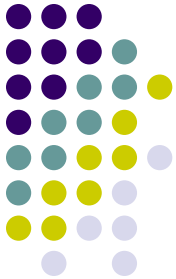
- Доступ к содержимому пакета дейтаграммы осуществляется с помощью методов:
- метод **getData(),** который возвращает массив байт, представляющий собой содержимое дейтаграммы. С другой стороны,
- методы **setData()** позволяют записать в пакет данные, получая в качестве параметров байтовые массивы.

Методы DatagramPacket



- Для экземпляра класса DatagramPacket определена группа методов, информирующих и управляющих работой дейтаграммы.. К наиболее используемым методам данного класса можно отнести следующие:
- **getAddress() и setAddress()-** возвращает / устанавливает адрес подключения;
- **getPort() и setPort()** – возвращает / устанавливает порт подключения;
- **getLength() и setLength()** – возвращает / устанавливает размер дейтаграммы.

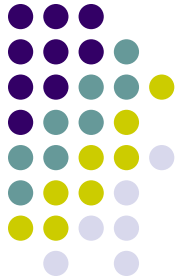
Исключения



- В процессе работы с объектами дейтаграмм могут возникать исключительные ситуации, для чего нужно реализовать соответствующую обработку следующих исключений:
- **SocketException** – ошибка протокола при обращении к сокету;
- **SecurityException** – ошибка доступа к системе безопасности;
- **UnknownHostException** – неправильно указан адрес InetAddress.

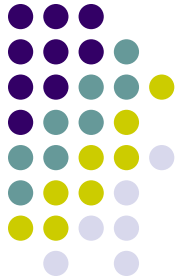
При завершении работы с сетевыми соединениями DatagramSocket его следует закрыть, используя метод **close()** данного класса.

Пример создания дейтаграмм

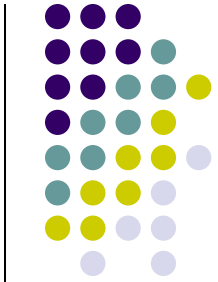


- Рассмотрим пример, иллюстрирующий использование дейтаграмм. В представленных программах одно приложение выступает в роли сервера и ожидает поступления на порт 8008 пакетов дейтаграмм. Если таковые были получены, то выполняется их преобразование в строковый эквивалент и полученный таким образом текст печатается на экран.
- Приложение-сервер опрашивает порт 8008 компьютера для получения дейтаграммы. После этого содержимое дейтаграммы выводится на экран.

Программа-сервер

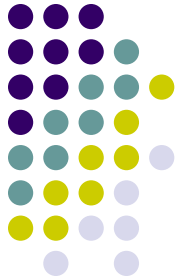


```
import java.net.*;
public class DatagramServer extends Thread
{
  byte[] buf = new byte[100];
  int port=8008;
  public void run()
   {
     try
       {
         // определяем локальный адрес
         InetAddress local=InetAddress.getByName("127.0.0.1");
         // определяем сокет дейтаграммы
         DatagramSocket srvsocket=new
    DatagramSocket(port,local);
```

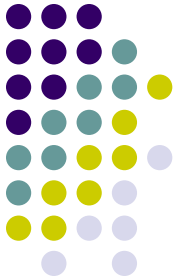


```
// определяем пакет дейтаграммы
        DatagramPacket pack=new
    DatagramPacket(buf,buf.length);
        while (true)
        {
          //получаем дейтаграмму
          srvsocket.receive(pack);
          // конвертируем ее в строку
          String str =new String(pack.getData(),0,pack.getLength());
          System.out.println(str);
        }
      }
      catch(Exception e)
      {
        System.out.println(e.getMessage());
      }
  }
```

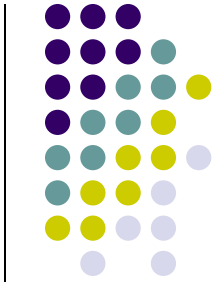
```
public static void main(String arg[])
  {
    System.out.println("Datagram server running");
    DatagramServer my=new DatagramServer();
    my.start();
  }
}
```



Программа-клиент

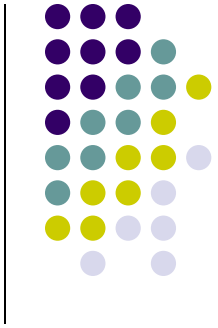


```
import java.net.*;
public class DatagramClient
{
 public static void main(String arg[])
{
// объявляем массив байт для пакета дейтаграммы
 byte[] buf=new byte[100];
//инициализация порта
 int port=8008;
 int cur=0;
 int ch;
 boolean flag=true;
```



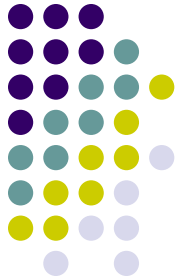
```
 while (flag)
 {
  try
    {
// получаем символ с клавиатуры
      ch=System.in.read();
// определяем локальный адрес
      InetAddress local=InetAddress.getLocalHost();
// записываем символ в массив
      buf[cur]=(byte)ch;
      if (ch==-1) flag=false;
      if (ch=='\n')
         {
// определяем сокет дейтаграмм
           DatagramSocket srvsocket=new DatagramSocket();
// определяем пакет дейтаграммы  для отправки по адресу
    local и порту port
           DatagramPacket pack=new
    DatagramPacket(buf,buf.length, local, port);
```

```
//  посылаем пакет через сокет
          srvsocket.send(pack);
      cur=-1;
          for (int i=0; i<buf.length;i++) buf[i]=0;
        }
      cur++;
      }
      catch(Exception e)
      {
        System.out.println(e.getMessage());
        break;
      }
    }
}
}
```



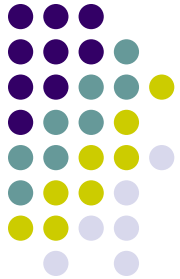
UDP-сокеты в C#

Фрагмент программы, ожидающей поступления дейтаграмм

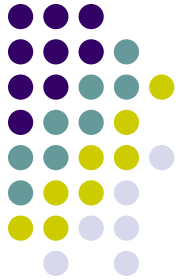


```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

public class UDPMulticastListener
{

    private static readonly IPAddress GroupAddress =
     IPAddress.Parse("127.0.0.1");
    private const int GroupPort = 13000;
```

Метод чтения дейтаграммы

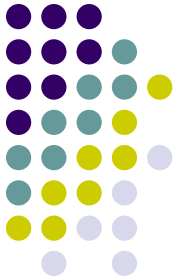


```
private static void StartListener()
    {
     bool done = false;
     UdpClient listener = new UdpClient(13000);
     IPEndPoint groupEP = new
    IPEndPoint(GroupAddress,GroupPort);
try
    {
```



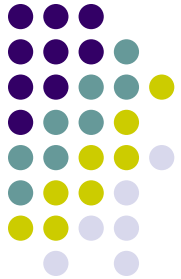
```
while (!done)
    {
      Console.WriteLine("Waiting for broadcast");
      byte[] bytes = listener.Receive(ref groupEP);
      Console.WriteLine("Received broadcast from {0} :\n {1}\n",
groupEP.ToString(),Encoding.ASCII.GetString(bytes,0,bytes.Len
    gth));
    }

    listener.Close();

}
catch (Exception e)
{
    Console.WriteLine(e.ToString());
}
}
```

Вызов метода



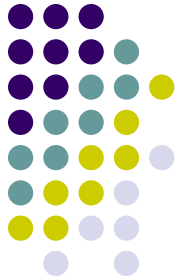
```
public static int Main(String[] args)
    {
     StartListener();


     return 0;
    }
}
```

Фрагмент программы, посылающей дейтаграмму



```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
public class UDPMulticastSender
{
private static IPAddress GroupAddress =
    IPAddress.Parse("127.0.0.1");
private static int GroupPort = 13000;
```

Метод, отправляющий дейтаграмму



```
private static void Send( String message)
{
    UdpClient sender = new UdpClient();
    IPEndPoint groupEP = new
    IPEndPoint(GroupAddress,GroupPort);
    try
    {
     byte[] bytes = Encoding.ASCII.GetBytes(message);
     sender.Send(bytes, bytes.Length, groupEP);
           sender.Close();
     Console.WriteLine(" Sending datagram : {0}", message);

    }
```

Вызов метода

```
catch (Exception e)
     {
          Console.WriteLine(e.ToString());
     }

    }

public static int Main(String[] args)
    {
     Send("text");
     Console.WriteLine("Sending datagram");
     return 0;
    }
}
```