Lecture 7

- ConnectivityManager, NetworkInfo
- URL, URLConnection, HttpURLConnection
  Downloading pictures, uploading text files
- JSON, XML
- Streams, Object Streams, Serializable
- UDP, TCP, Network Communications in Android

ConnectivityManager, NetworkInfo

To obtain information about the network status you must add a permission in the AndroidManifestet: ACCESS_NETWORK_STATE

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Accessing the network also requires a permission declared in the AndroidManifestet: INTERNET

```
<uses-permission android:name="android.permission.INTERNET" />
```

The class Connectivitymanager provides information on the status of the network connection.

The class NetworkInfo provides information about the status of specific networks, i.e. mobile broadband and/or Wi-Fi.

```
ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE); // Context

NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
if(networkInfo!=null) {
    if(networkInfo.isAvailable())
        // Network is available

    if(networkInfo.isConnected())
        // Device is connected
}
```

ConnectivityManager, NetworkInfo

A method for controlling the connection:

```
public boolean isOnline(Context context) {
    ConnectivityManager connMgr =(ConnectivityManager)
        context.getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo info = connMgr.getActiveNetworkInfo();
    return (info!=null && info.isConnected());
}
```

It is possible to obtain the status of the connection to the mobile network or Wi-Fi.

Information about mobile connectivity.

```
networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
```

Information about Wi-Fi.

```
networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
```

AsyncTask

All network operations must be performed by a separate thread (not the UI thread). AsyncTask provides the appropriate methods to connect to a server and to retrieve resources.

```
public class LoadTask extends AsyncTask<String,Integer,Task> {
    private TaskListener listener;

    public LoadTask(TaskListener listener, ...) {
        if(listener!=null) {
            this.listener = listener;
            execute(...); // Thread started. Continues in doInBackGround
        }
    }

    protected Task doInBackground(String... params) {
        Task result = null
                // Get the job done here
        return result;
    }

    protected void onPostExecute(Task result) {
        listener.taskLoaded(result);
    }

    public interface TaskListener {
        public void taskLoaded(Task result);
    }
}
```

Downloading a bitmap

```
public class LoadBitmap extends AsyncTask<String,Integer,Bitmap> {
    private BitmapListener listener;

    public LoadBitmap(BitmapListener listener, String url) {
        if(listener!=null) {
            this.listener = listener;
            execute(url);
        }
    }

    protected Bitmap doInBackground(String... params) {
        Bitmap result = null;
        InputStream input=null;
        try {
            URL url = new URL(params[0]);
            URLConnection connection = url.openConnection();
            input = connection.getInputStream();
            result = BitmapFactory.decodeStream(input);
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // close inputstream!  --------->
        }
        return result;
    }

    public void onPostExecute(Bitmap result) {…}

    public interface BitmapListener {
        public void bitmapLoaded(Bitmap bitmap);
    }
}
```

```
try {
    if(input!=null)
        input.close();
} catch(IOException e) {}
```

Downloading a bitmap

Create your own class extending BitmapListener:

```
// inner class
private class BL implements LoadBitmap.BitmapListener {
    public void bitmapLoaded(Bitmap bitmap) {
        if( bitmap!=null )
            imView.setImageBitmap(bitmap);
    }
}
```

Then you create an object of type LoadBitmap.

```
new LoadBitmap(new BL(),
               "http://ddwap.mah.se/tsroax/memory/Memory.jpg");
```

When the bitmap is loaded or something has gone wrong, bitmapLoaded in BitmapListener will be called.

Downloading a text file

```
public class LoadText extends AsyncTask<String,Integer,String> {
    private TextListener listener;

    public LoadText(TextListener listener, String url, String encoding) {
        if(listener!=null) {
            this.listener = listener;
            execute(url, encoding);
        }
    }

    protected String doInBackground(String... params) {
        StringBuilder result = new StringBuilder();
        BufferedReader reader = null;
        try {
            URL url = new URL(params[0]);
            URLConnection connection = url.openConnection();
            reader = new BufferedReader(new InputStreamReader(
                            connection.getInputStream(), params[1]));
            String txt;
            while((txt=reader.readLine())!=null)
                result.append(txt+"\n");
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // close reader
        }
        return result.toString();
    }

    protected void onPostExecute(String result) {…}

    public interface TextListener {…}
}
```

Downloading a text file

Create your own TextListener that overrides textLoaded:

```
new LoadText(new LoadText.TextListener() {
    public void textLoaded(String str) {
        textView.setText(str);
    }
}, "http://ddwap.mah.se/tsroax/memory/Memory.html", "ISO-8859-1" );
```

It may be clearer to implement the TextListener as an inner class, as in the previous example:

```
private class TL implements LoadText.TextListener {
    public void textLoaded(String str) {
        textView.setText(str);
    }
}
```

And finally, instantiate LoadText:

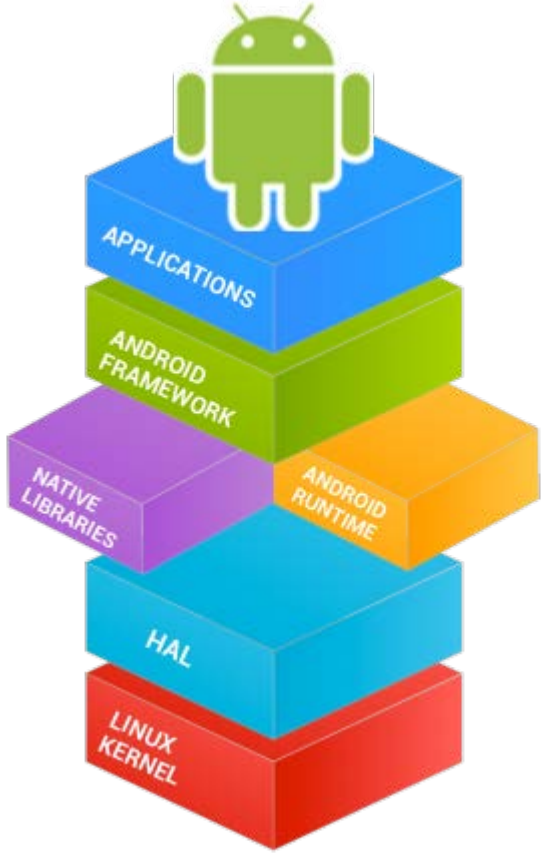
```
new LoadText(new TL(),
             "http://ddwap.mah.se/tsroax/memory/Memory.html",
             "ISO-8859-1" );
```

BONUS TRACK: using Volley for requesting data

- Volley is an HTTP library that makes networking for Android apps easier.
- **Very good** at handling multiple network connections.
- **Not suitable** for large download or streaming operations. For those, you use DownloadManager instead.
- To use Volley, add the following dependency to the build.gradle file:

```
compile 'com.android.volley:volley:1.0.0'
```

- Volley is distributed as a part of the Android Open Source Project (AOSP).



Source: https://developer.android.com/training/volley/index.html

BONUS TRACK: using Volley for requesting data

- To do a simple Volley GET request, first instantiate a **RequestQueue**, and then add a **StringRequest** instance to it.

```
private void volleyRequest() {

    // Instantiate the RequestQueue.
    RequestQueue queue = Volley.newRequestQueue(this);

    String key = " here you place your API key ";
    String url = " here you place the URL for your query ";

    // Request a string response from the provided URL.
    StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {
                    processJSON(response);
                }
            }, new Response.ErrorListener() {
                @Override
                public void onErrorResponse(VolleyError error) {
                    Log.e("ERROR", error.getMessage(), e);
                }
            });

    // Add the request to the RequestQueue.
    queue.add(stringRequest);

}
```

BONUS TRACK: using Volley for requesting data

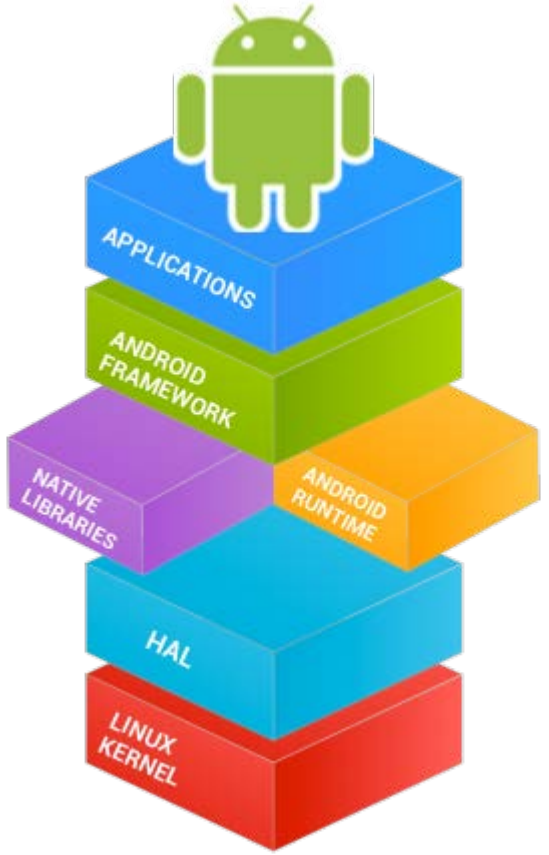
- A StringRequest needs:
  - The request method: GET, POST, etc. (find all of them in Request.Method)
  - The target URL.
  - A response listener that receives the answer to the request. Implement the interface Response.Listener<>.
  - An error listener that handles possible errors. Implement the interface Reponse.ErrorListener.

```
// Request a string response from the provided URL.
StringRequest stringRequest = new StringRequest(
          Request.Method.GET,
          url,
          new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                processJSON(response);
            }
          },
          new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                Log.e("ERROR", error.getMessage(), e);
            }
        });
```

BONUS TRACK: using Volley for requesting data

- For an extended use of Volley, read this:

https://developer.android.com/training/volley/index.html



JSON

JSON (JavaScript Object Notation) is a text format that is quite easy to read.

- **JSON-object**
  An object is bounded by **braces** {...}.

  An object consists of a number of key:value pairs. The key and the value are enclosed in **quotes** and separated by **colons**.

```
{"firstName":"Anna", "lastName":"Nilsson"}
```

- **JSON-array**
  The value of a key:value pair can be an array.

  The array is enclosed by **brackets** [...].

  The array consists of a number of **comma** separated items. Every item is also a key:value pair. The key and the value are enclosed in **quotes** and separated by **colons**.

```
{"persons" : [{"firstName":"Anna", "lastName":"Nilsson"},
              {"firstName":"Nils", "lastName":"Konti"},
              {"firstName":"Kim", "lastName":"Dolan"}]}
```

Reading JSON

To read the contents of a JSON-formatted file, you need the file as a string.

Create an InputStream from the source:

```
url = new URL(”http://...");
HttpURLConnection connection = (HttpURLConnection)url.openCon...();
InputStream input = connection.getInputStream();
```

Convert it to String:

```
String str = new Scanner(input, ”UTF-8”).useDelimiter(”\\A”).next();
```

Create a JSON object out of the String

```
JSONObject json = new JSONObject(str);
```

JSONObject /JSONArray

To read values from a JSONObject uses getters:

- getJSONObject(name)
- getJSONArray(name)
- getString(name) / getInt(name), etc.

You can iterate a JSONArray element by element with the length() method, together with getAAA(index) methods.

```
JSONOBJECT transaction;
JSONObject json = …;
JSONArray array = json.getJSONArray(“transaction”);
for(int i=0; i<array.length(); i++) {
    transaction = array.getJSONObject( i );
    dateString = transaction.getString(“date”);
    // etc
}
```

JSON - an example

```
{”transaction":[
     { "date": "130923", "type", "food", "amount": "314"},
     { "date": "130923", "type", "trip", "amount", "39.20"},
     { "date": "130923", "type", "food", "amount": "60"},
     { "date": "130923", "type", "leisure", "amount": "92"},
     { "date": "130923", "type", "leisure", "amount": "70"},
     { "date": "130924", "type", "rent", "amount": "4200"},
     { "date": "130924", "type", "trip", "amount", "78.40"},
     { "date": "130924", "type", "miscellaneous", "amount": "140"},
     { "date": "130924", "type", "PM \ u00e4der", "amount": "420"},
     { "date": "130924", "type", "food", "amount": "65"},
     { "date": "130924", "type", "leisure", "amount": "35"}]}
```

Transaction
-date:String -type:String -amount:String
+Transaction(String,String,String)
+getDate() : String +getType() : String +getAmount() : String

TableLayout

Transaction objects are placed in rows in a TableLayout.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/..."
    android:id="@+id/table“
    android:layout_width="match_parent“
    android:layout_height="wrap_content“
    android:scrollbars="vertical" >
</TableLayout>
```

Each row in the table is defined by a layout:

```
<?xml version="1.0" encoding="utf-8"?>
<TableRow xmlns:android="http://schemas.android.com/apk/res/android" >
    <TextView
        android:id="@+id/tvDate"
        android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="wrap_content" />
    <TextView
        android:id="@+id/tvType"
        android:layout_weight="3"... />
    <TextView
        android:id="@+id/tvAmount"
        android:layout_weight="1"
        android:gravity="right"... />
</TableRow>
```

TableLayout

```
private TableLayout table;
:
private class JL implements JsonListener {
    private void putRow(LayoutInflater inflater, String date, String type,
                        String amount, int background, int textColor) {
        TextView tvDate, tvType, tvAmount;
        View view = inflater.inflate(R.layout.tablerow, null);
        tvDate = (TextView)view.findViewById(R.id.tvDate);
        tvType = (TextView)view.findViewById(R.id.tvType);
        tvAmount = (TextView)view.findViewById(R.id.tvAmount);
        tvDate.setText(date);
        tvDate.setBackgroundColor(background);
        tvDate.setTextColor(textColor);
        // setters for tvType and tvAmount
        table.addView(view);
    }

    public void jsonLoaded(ArrayList<Transaction> list) {
        Activity activity = JsonTable.this.getActivity();
        LayoutInflater inflater = activity.getLayoutInflater();
        putRow(inflater,"Datum","Post","Belopp",Color.BLACK,Color.WHITE);
        for(Transaction tr : list) {
            String twoDec = String.format("%1.2f",
                                Double.parseDouble(tr.getAmount()));
            putRow(inflater,tr.getDate(),tr.getType(),
                   twoDec,Color.LTGRAY,Color.BLACK);
        }
    }
}
```

XML

XML stands for Extensible Markup Language and it is another text format that is quite easy to read.

- **Element**
  An element consists of the start tag and the end tag

```
<element>
:
</element>
```

  or of a blank tag:

```
<element />
```

- **Attribute**
  An element can contain attributes

```
<element id="18227345" type="randig"> ... </element>
<element id="18227345" type="randig“ />
```

- An element may contain other elements and text

```
<book>
    <author> Rolf </author>
    <author> Anna </author>
    <isbn nbr="748376455-3"> ... </isbn>
</book>
```

XmlPullParser

You can use a XmlPullParser to read the contents of an XML file.

You may need an InputStrem as well:

```
url = new URL(”http://...");
HttpURLConnection connection = (HttpURLConnection)url.openCon...();
InputStream input = connection.getInputStream();
```

Then create a XmlPullParser object and initialize it:

```
XmlPullParser parser = Xml.newPullParser();
parser.setInput(new InputStreamReader(input, "UTF-8"));
parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);
```

When the scanning is finished, close the input stream:

```
input.close();
```

XmlPullParser

The parser can be in one of the following five states (state):

- **START_DOCUMENT**
  At the beginning of the document
- **END_DOCUMENT**
  At the end of the document
- **START_TAG**
  A start tag just loaded.
  The method getName () returns the name of the element.

  The method getAttributeValue (null, ID) gives the value of an attribute
- **END_TAG**
  An end tag just loaded. The method getName() returns the name of the element.
- **TEXT**
  Text just loaded. The method getText () returns the text.

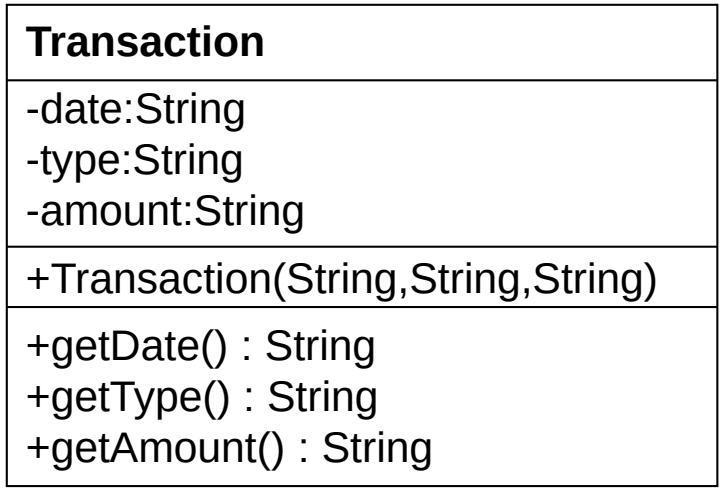
XmlPullParser

You can parse the whole file within a while loop:

```
int eventType = parser.getEventType();
while( eventType!=XmlPullParser.END_DOCUMENT ) {
    switch( eventType ) {
        case XmlPullParser.START_DOCUMENT :
            // initialize
        case XmlPullParser.START_TAG :
            // handle every element and read every attribute
        case XmlPullParser.TEXT :
            // Read a text field
        case XmlPullParser.END_TAG :
            // Gather all values parsed and create an object out of
         // them
    }
    eventType = parser.next();
}
```

XmlPullParser – an example

```
<transaction>
    <expenditure date = "130 923" type = "food" amount = "314" />
    <expenditure date = "130 923" type = "travel" amount = "39.20" />
    <expenditure date = "130 923" type = "food" amount = "60" />
    <expenditure date = "130 923" type = "recreational" amount = "92" />
    <expenditure date = "130 923" type = "recreational" amount = "70" />
    <expenditure date = "130 924" type = "rent" amount = "4200" />
    <expenditure date = "130 924" type = "travel" amount = "78.40" />
    <expenditure date = "130 924" type = "diverse" amount = "140" />
    <expenditure date = "130 924" type = "clothes" amount = "420" />
    <expenditure date = "130 924" type = "food" amount = "65" />
    <expenditure date = "130 924" type = "recreational" amount = "35" />
</ transactions>
```



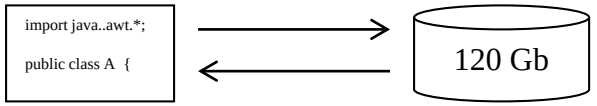
XmlPullParser – an example

The method readTransactions parses the file andcreates a Transaction object per every <expenditure> tag found. Transaction objects are stored in an ArrayList which is returned by the method.

```
private ArrayList<Transaction> readTransactions(XmlPullParser parser)
                throws IOException, XmlPullParserException {
    ArrayList<Transaction> list = new ArrayList<Transaction>();
    Transaction transaction;
    String name, date, type, amount;
    int eventType =   parser.getEventType();
    while(eventType != XmlPullParser.END_DOCUMENT) {
        if(eventType==XmlPullParser.START_TAG) {
            name = parser.getName();
            if(name.equals(“expenditure")) {
                date = parser.getAttributeValue(null, “date");
                type = parser.getAttributeValue(null, "type");
                amount = parser.getAttributeValue(null, “amount");
                list.add(new Transaction(date,type,amount));
            }
        }
        eventType = parser.next();
    }
    return list;
}
```

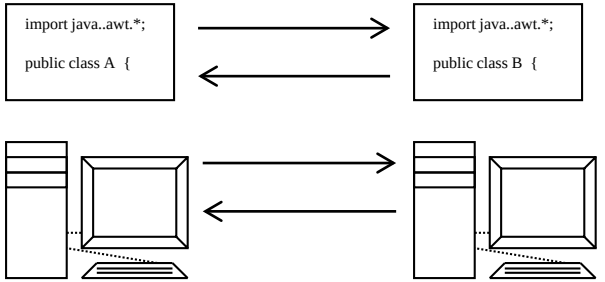
Streams

You often need to move data between
from memory to the hard drive



between two programs
on the same computer

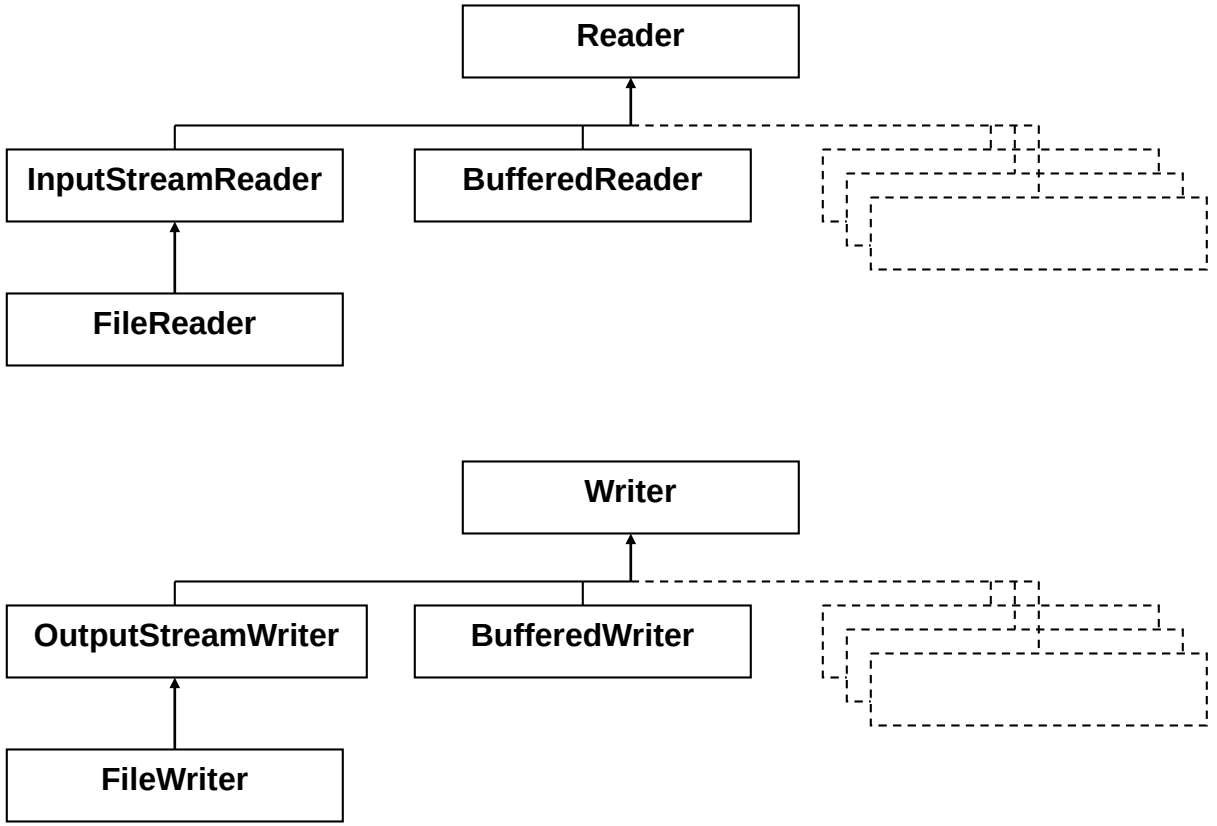
or running on different computers



Streams are data flows between a source and a target, often from an application to different devices, such as a disk file or a program on another computer.
If the flow to go in both directions, it requires two streams. The flow off the program is called the output stream ("write data"), and the opposite is called the input stream ("reading data").

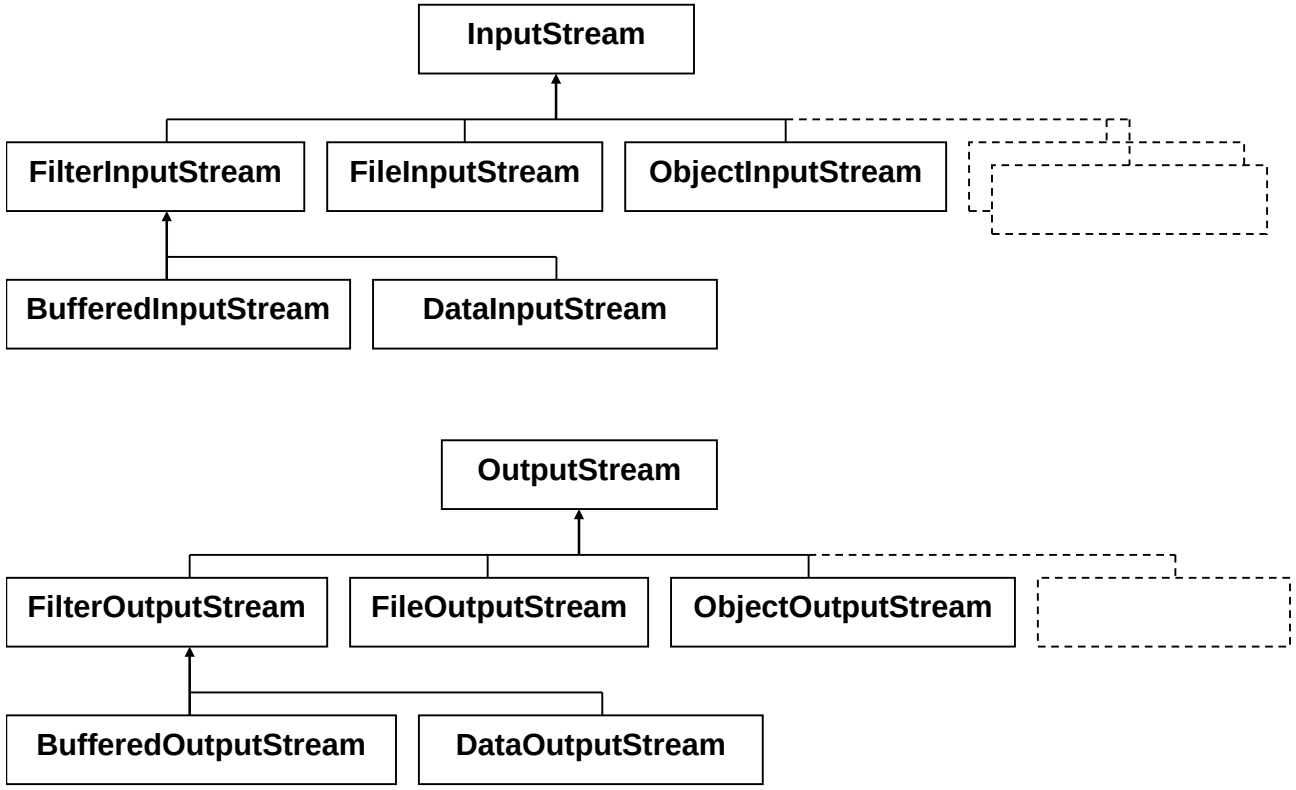
Text streams

The basic classes for reading and writing text are Reader and Writer. Both classes are abstract.



Data streams

The basic classes for implementing data streams are InputStream and OutputStream . Both classes are abstract.



Output Data streams

OutputStream contains, among others, the following methods

public void close()	Close the stream
public void flush()	Flush the buffer

DataOutputStream implements DataOutput and thus contains, among others, these methods:

public void writeBoolean(boolean)	Write a boolean
public void writeByte(byte)	Write a byte
public void writeChar(char)	Write a char
public void writeChars(String)	Write a String
public void writeDouble(double)	Write a double
public void writeFloat(float)	Write a float
public void writeInt(int)	Write an int
public void writeLong(long)	Write a long
public void writeShort(short)	Write a short
public void writeUTF(String)	Write an UTF-coded String

All the above methods can throw IOExceptions.

Input Data streams

InputStream has, among others, the following method:

public void close()	Close the stream

DataInputStream implements **DataInput**, so i has, among others, the following methods:

public boolean readBoolean()	Reads a boolean
public byte readByte()	Reads a byte
public char readChar()	Reads a char
public double readDouble()	Reads a double
public float readFloat()	Reads a float
public int readInt()	Reads an int
public long readLong()	Reads a long
public short readShort()	Reads a short
public String readUTF()	Reads a UTF-coded String

All the above methods can throw IOExceptions.

Writing objects

Create a FileOutputStream:

```
FileOutputStream fos = new FileOutputStream(”C:/ex.dat”);
```

Connect this FileOutputStream to an ObjectOutputStream:

```
ObjectOutputStream oos = new ObjectOutputStream(fos);
```

Write the object using:

```
public void writeObject(Object obj)
```

```
public void SaveToFile(String filnamn) throws IOException {
  ObjectOutputStream oos = new ObjectOutputStream(new FileOut…));
  oos.writeObject(getText());
  oos.close();
}
```

The object that is written must be an instance of a class that implements the interface Serializable or Externalizable.

Reading objects

Create a FileInputStream:

```
FileInputStream fis = new FileInputStream(”C:/ex.dat”);
```

Connect it with an **ObjectInputStream**:

```
ObjectInputStream ois = new ObjectInputStream(fis);
```

Read the object using:

```
public Object readObject()
```

Which will return a reference to an object that should be casted to its actual class.

```
public void ReadFromFile(String filnamn) throws IOException {
  ObjectInputStream ois = new ObjectInputStream(new FileInputS..));
  try {
    setText((String)ois.readObject());          // Cast to String
  }
  catch(ClassNotFoundException e) {}
  ois.close();
}
```

The object must be an instance of a class that implements the interface Serializable or Externalizable.

Serializable

Class Person implements Serializable. Instances from Person can be written to / read from a file.

```
import java.io.*;

class Person implements Serializable {
   private String name;
   private Person partner;
```

Write an array of Persons.

```
public static void writePersons(Person[] pers) throws IOException {
   ObjectOutputStream oos = new ObjectOutputStream(new FileOut…);
   oos.writeObject(pers);
   oos.close();
}
```

Read an array of Persons.

```
public static Person[] readPersons() throws IOException {
   Person[] pers=null;
   ObjectInputStream ois = new ObjectInputStream(new FileInputS…);
   try {
      pers = (Person[])ois.readObject();
   }
   catch(ClassNotFoundException e) {}
   ois.close();
   return pers;
}
```

Datagram - UDP

One can communicate with datagrams over the network. But this communication does not ensure that:

- the content is preserved while its being transported,
- the datagram reaches the receiver,
- many datagrams will arrive in the same order in which they were sent.

A computer sends a "package", which contains the sender address, the recipient address, and any kind of data.

The recipient receives the package. Communication takes place over the defined port numbers.

Both the sender and the recipient need a DatagramSocket and a DatagramPacket.

The package is sent with the Send method, and it is received with the receive method.

Datagram

Sender:

```
InetAddress      till = InetAdress.getByName(”123.231.111.123”);
DatagramSocket   socket = new DatagramSocket();
DatagramPacket   packet;
String           message=”Meddelande att skicka”;
:
byte [] data = message.getBytes();
packet = new DatagramPacket(data, data.length,till,2345);
socket.send(packet);
:
socket.close();
```

Recipient (ip = 123.231.111.123)

```
DatagramSocket   socket = new DatagramSocket(2345);
byte[]           data = new byte[256];
DatagramPacket   packet = new DatagramPacket(data, data.length);
String           message;
:
socket.receive(packet);
message = new String(packet.getData(), 0, packet.getLength());
:
socket.close();
```

InetAddress

Internet addresses and IP addresses are described in the class InetAddress.

Class methods (throws UnknownHostException)

public static InetAddress[] getAllByName(String host)
Returns all IP addresses on the specified computer as an InetAddress array.
public static InetAddress getByName(String host)
Returns an Inet Address to the specified computer.
public static InetAddress getLocalHost()
Returns your local InetAddress.

Useful methods

public String getHostAddress()
Returns the IP address in the form "123205121123"
public String getHostName()
Returns the host name

Online connection - TCP / IP

If you have the need for increased security in communication between computers, you can use the classes ServerSocket and Socket.

The server uses both ServerSocket and Socket while clients only need to use Socket.

Communication takes place through a stream in each direction, i.e. through an InputStream and a FileOutputStream.

For transferring simple data types + strings you can connect streams to DataInputStream and DataOutputStream.

Otherwise, you can connect streams to ObjectInputStream and ObjectOutputStream .

ServerSocket and Socket

ServerSocket

public ServerSocket(int port) throws IOException
Returns a ServerSocket listening in that port.

public Socket accept() throws IOException
Returns the socket created by the client. This method waits until a client wants to join.

public void close() throws IOException
Closes the ServerSocket

Socket

public Socket(String servernamn, int port) throws IOException
Creates a socket that connects the port on the specified server.

public Socket(InetAddress server, int port) throws IOException
Creates a socket that connects the port on the specified server.

public void close() throws IOException
Closes the socket.

public InputStream getInputStream() throws IOException
Returns a stream for reading data from the other computer.

public OutputStream getOutputStream() throws IOException
Returns a stream to send data to the other computer.

Network communication in Android

1. For network communications, Android can’t do all the related operations in the UI thread. So you have to create thread/s to handle the communication. This way, the UI thread will not be blocked by the network communication.

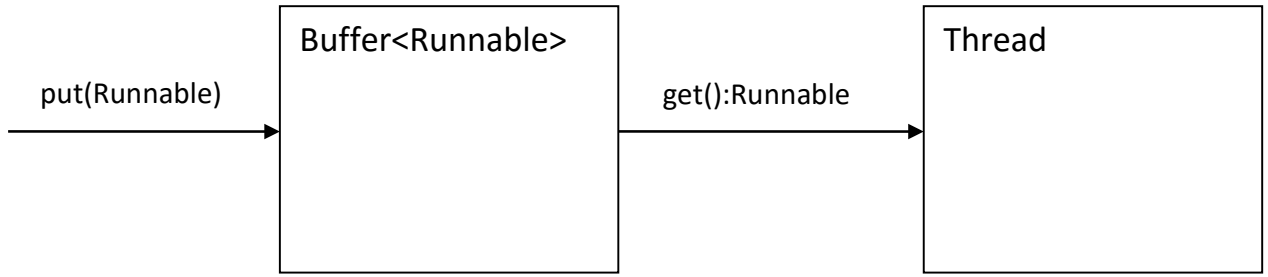
2. When the configuration is changed on the device, the system restarts the app, making communications more difficult. This can be handled by using the class Service.

It is also possible to use retained fragments.

Using separate thread

The class RunOnThread is an example of how to use a thread to execute instructions. The class consists of two parts: a buffer and a worker.

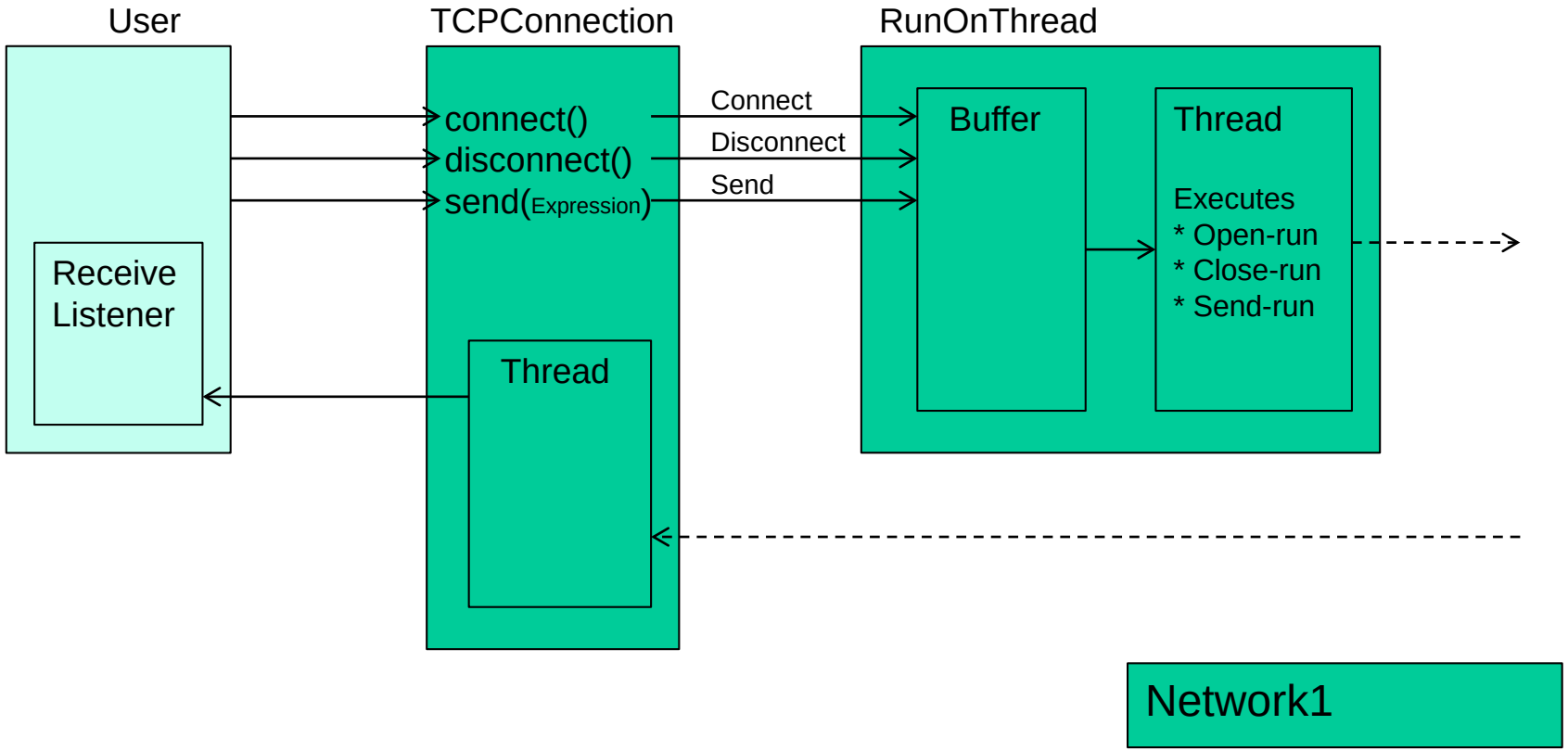
The user places the appropriate items in the buffer (put method). Then the thread retrieves the items (get method) and executes the code in them.



RunOnThread

Design of a TCP-client

The project TCPWithoutService shows an example of communication with TCP. The example does not use Service . The design looks like this:



Service – start / finish

An instance of the service is retained even if the Activity that started the service is terminated. This makes it quite useful for processes that should not be interrupted when, for instance, the device is rotated.

The method startService(intent) instantiates a Service. You can call onStartCommand from a Service instance. The intent can transfer important information to the Service, such as the IP address of the server you should connect to.

Method stopService(intent) ends the service.

Service – bind/unbind, IBinder/Binder,…

Communicate with the server from the application by calling the method to bindService(Intent, ServiceConnection, 0).

The second argument is a class that implements the Service Connection:

```
public interface ServiceConnection {
    public void onServiceConnected( String, IBinder );
    public void onServiceDisconnected( String );
}
```

In Service, the method onBind(Intent) is executed. The method returns an instance of a class that implements an IBinder.

```
public IBinder onBind(Intent intent) {
    return new LocalService();
}
public class LocalService extends Binder {  // Binder extends IBinder
    public TCPConnection getService() {
        return TCPConnection.this;   // reference to the Service
    }
}
```

Service – bind/unbind, IBinder/Binder,…

If onBind returns null, the binding was unsuccessful.

The method onServiceConnected (..., Binder) will be called by the system. The second argument in the call, a Binder, is the object that is returned by onBind.

```
private class ServiceConn implements ServiceConnection {
    public void onServiceConnected(ComponentName arg0, IBinder binder) {
        TCPConnection.LocalService ls = (TCPConnection.LocalService) binder;
        connection = ls.getService(); // connection holds the reference to the Service
        :
    }
}
```

Now, connection holds a reference to the service object.

unbindService(ServiceConnection) – Called before the app is terminated.

TCPService

Using a Service the app will not lose any information after the device is, for instance, rotate. Service stores all the incoming data in a buffer. The application retrieves the stored values when active by means of a wire. The design is analogous to the former one.