1. Adaugă JSoup-1.8.2.jar în libs, Build Path, Add to Build Path (apare Referenced Libraries)
2. Implementarea intefeţei grafice:
    a. în xml-ul activity main se definesc elementele grafice
    b. în xml-ul string din res/values se pun şirurile de caractere asociate elementelor grafice
    c. pentru spinnere sunt necesare stringuri de forma:

```
<string-array name="information_types">

        <item>temperature</item>

        <item>wind_speed</item>

        <item>condition</item>

        <item>pressure</item>

        <item>humidity</item>

        <item>all</item>

</string-array>
```

    d. se declară global în Main Activity elementele grafice şi se instanţiază pe metoda on create utilizând metoda findViewById(R.id.<id-elem>)

3. Activare permisiune INTERNET:
   <uses-permission
     android:name="android.permission.INTERNET" />

4. Creare server
    a. reţine în variable globale port-ul, ServerSocket şi structura de reţinere date (ex Hashmap - în acest punct se definesc eventualele clase care definesc tipul de date reţinut). Constructorul va primi ca parametru doar portul şi va crea instanţe de ServerSocket şi de date)
    b. pe metoda run se acceptă conexiuni socket = serverSocket.accept()
    c. se crează o instanţă a thread-ului de comunicaţie server -client
    d. se porneşte thread-ul de comunicaţie communicationThread.start()
    e. se deduce faptul că thread-ul de comunicaţie trebuie să reţină serverThread-ul şi socket-ul clientului (cel preluat de metoda .accept()) => se poate crea o clasa noua care extinde clasa Thread numită CommunicationThread care să conţină o variabilă globală pentru ServerThread şi o variabilă de tip Socket; Totodată se

poate crea și constructorul care asociază obiectele primite ca parametrii cu variabilele globale

f. oprirea ServerThread-ului: într-o metodă din cadrul serverului se apelează metoda interrupt() care, cel mai probabil, întrerupe thread-ul curent și se închide socketul serverSocket.close(). În activitatea principală (cea unde se crează și rulează elementele grafice) se apelează în metoda onDestroy() serverThread().stopThread() - metoda proaspăt implementată. Se poate deduce astfel că în activitatea Main, în clasa asociată este necesară definirea variabilelor de tip ServerThread și ClientThread.

5. Creare thread comunicatie
    a. am dedus anterior că această clasă extinde Thread și reține cel puțin serverThread și socket client
    b. acest tip de Thread se va remarca doar pe metoda run() unde va instanția cele două tipuri de stream-uri pe care le va realiza cu clientul (-> <-); În acest punct se va definini clasa Utilities ca wrapper peste funcțiile de instanțiere stream-uri de comunicație

```
BufferedReader bufferedReader = Utilities.getReader(socket);
PrintWriter    printWriter    = Utilities.getWriter(socket);
```

    c. în acest thread se va analiza dacă serverul are informația cache-uită dintr-o interogare anterioară sau se va face un request => implementarea metodelor getData and setData pe thread-ul server. întrucât aceste metode pot fi apelate de mai multe thread-uri client simultan este necesară utilizarea keyword-ului **synchronized**. Datele de căutat vor fi preluate din cerea clientului => vor fi preluate de pe BufferReader (bufferReader.readLine())
    d. În cazul în care în structura de date preluată de la server pe metoda getData nu am găsit informația dorită, atunci se interoghează serverul care poate furniza date. Exemplu conexiune HTTP utilizând metoda post:

// - parte de initiere conexiune

```
HttpClient httpClient = new DefaultHttpClient();

HttpPost httpPost = new HttpPost(Constants.WEB_SERVICE_ADDRESS);

List<NameValuePair> params = new ArrayList<NameValuePair>();

params.add(new BasicNameValuePair(Constants.QUERY_ATTRIBUTE, city));

UrlEncodedFormEntity urlEncodedFormEntity = new UrlEncodedFormEntity(params, HTTP.UTF_8);

httpPost.setEntity(urlEncodedFormEntity);
//- primirea codului sursă al paginii html pentru parsare

ResponseHandler<String> responseHandler = new BasicResponseHandler();

String pageSourceCode = httpClient.execute(httpPost, responseHandler);
```

e. parsarea conţinutului paginii web primite ca răspuns al cererii de tip POST (necesită analiză, în prealabil, al codului sursă pentru a putea ştii unde se găseşte informaţia relevantă
   i. parsarea utilizând JSOUP:

```
Document document = Jsoup.parse(pageSourceCode);
Element element = document.child(0);
Elements scripts =
Element.getElementsByTag(Constants.SCRIPT_TAG);
for (Element script: scripts) { //iterare elemente care contin tagul specificat
în Constants.SCRIPT_TAG
    String scriptData = script.data();
}
```

   ii. parsarea utilizând JSON

```
JSONObject content = new JSONObject(scriptData);
                JSONObject currentObservation =
content.getJSONObject(Constants.CURRENT_OBSERVATION);
String temperature =
currentObservation.getString(Constants.TEMPERATURE);
```

f. Thread-ul de comunicatie va prelua informatia nou descoperită şi o va cache-ui pe Thread-ul server. În acest moment informaţia se va transmite cu metoda implementată la subpunctul c serverThread.setData(). Cel mai probabil va fi necesară instanţierea unei clase de tip structura de date. (ex: o clasă care reţine informaţii despre vreme). În acest moment informaţia este persistentă pe server, dar clientul încă nu a primit informaţia solicitată.
g. Pentru a transmite informaţia către client se va utiliza stream-ul de scriere (printWriter)

```
printWriter.println(result);
printWriter.flush();
```

   Abia după flush se va transmite informaţia
h. După transmiterea informaţiei se va închide socketul prin socket.close()

6. Creare thread client
   a. se vor defini global atât datele de conectare către server (adresă + port) cât şi informaţia de stocat sau de query. Acestea vor fi transmise de interfata grafica, din câmpurile completate de utilizatorul aplicaţie. (Se va ţine minte faptul că vor trebui declanşate pornirea server-ului şi clientului => în interfaţa grafică vor trebui generate listeneri)

b. pe metoda run se vor initializa stream-urile de ieşire şi intrare (aceleaşi operaţii ca în cazul Thread-ului de comunicaţie) nu înainte de a se crea socket-ul.
c. Clientul va trimite Thread-ului de comunicaţie informaţia necesară (printWriter.println(string); printWriter.flush())
d. De pe socketul de primire (stream-ul BufferReader) se va citi linie cu linie, cât timp va returna date. În cazul în care informaţia se doreşte a fi scrisă în interfaţa grafică, acest fapt nu se poate face direct întrucât operaţiile de networking pot fi blocante. Din acest motiv, global va fi definită şi structura din interfaţa grafică în care se va scrie, primindu-se referinţă prin constructor. În momentul în care se face scrierea, pe componenta grafică se apelează metoda post(new Runnable {...}) ex:

```
final String finalizedWeatherInformation = weatherInformation;
  weatherForecastTextView.post(new Runnable() {
        @Override
        public void run() {
        weatherForecastTextView.append(finalizedWeatherInformation +
"\n");
    }
  });
```

e. La finalizarea primirii datelor se închide socketul

7. Crearea de listeneri pe butonale care pornesc serverul şi clientul
    a. În clasa listener client se preiau datele din câmpurile editabile ale aplicaţiei, se instanţiază şi se rulează thread-ul client. A se asocia listener-ul cu elementul grafic! (idem pentru server)
8. Testare clienţi multiplii
    a. nc <adresa ip server> <port> : se vor scire în ordine datele aşa cum se aşteaptă thread-ul de comunicaţie să le preia de pe stream-ul de input