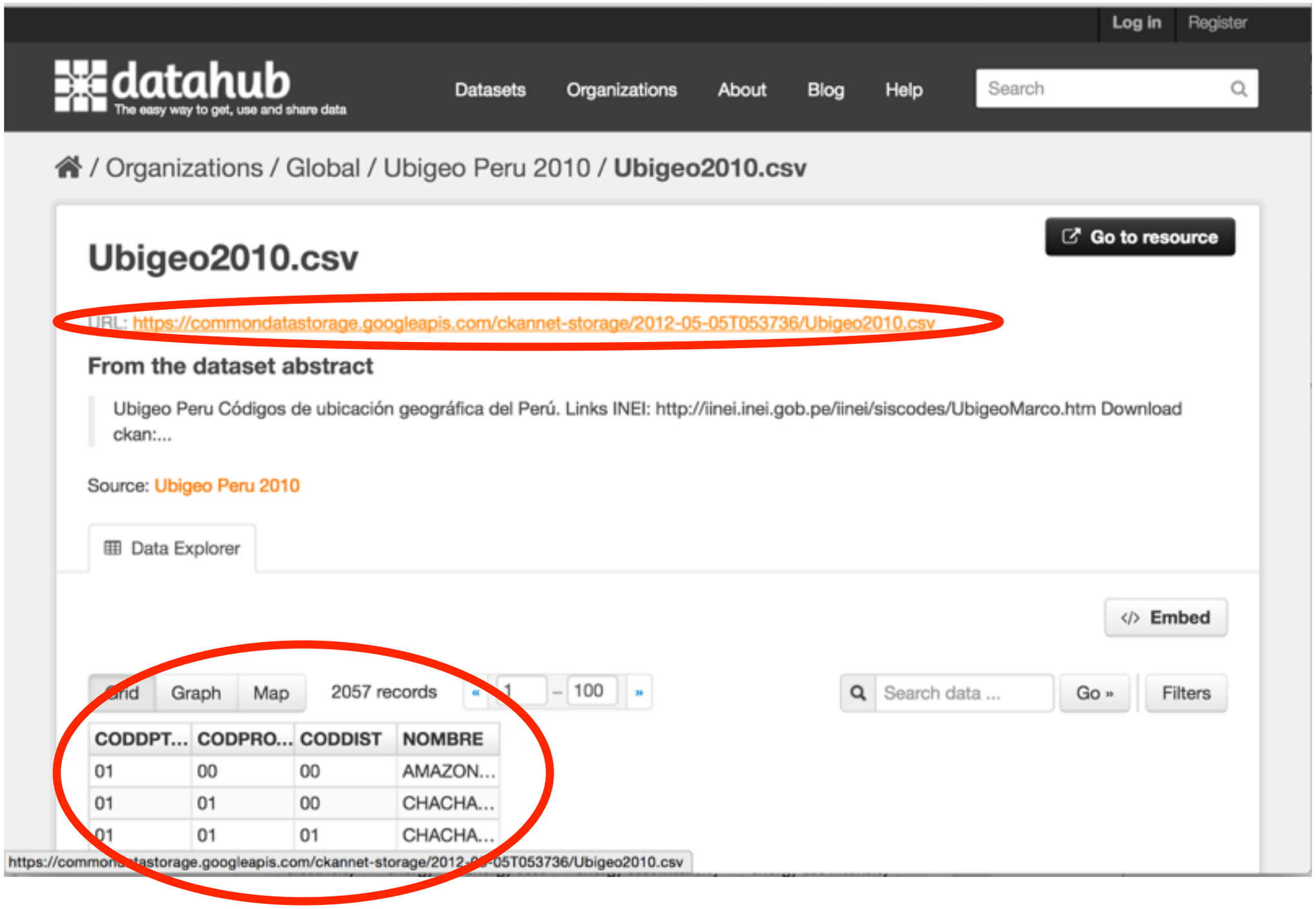
A GENERIC FRAMEWORK FOR ENGAGING ONLINE DATA SOURCES IN INTRODUCTORY PROGRAMMING COURSES

NADEEM ABDUL HAMID

BERRY COLLEGE
Experience it Firsthand

"LIVE" DEMO

https://datahub.io/dataset/ubigeo-peru/resource/12c2cc3a-5896-496b-96f6-d95cd1618d61



CONNECT – LOAD – FETCH

```
Welcome to DrJava.  Working
> run PeruData1
2057
AREQUIPA
>
```

```
import core.data.*;

public class PeruData1 {
  public static void main(String[] args) {
    DataSource ds = DataSource.connect("https://..
    ds.load();
    String[] names = ds.fetchStringArray("NOMBRE")

    System.out.println(names.length);
    System.out.println(names[367]);
  }
}
```

WHAT'S IN THE DATA?

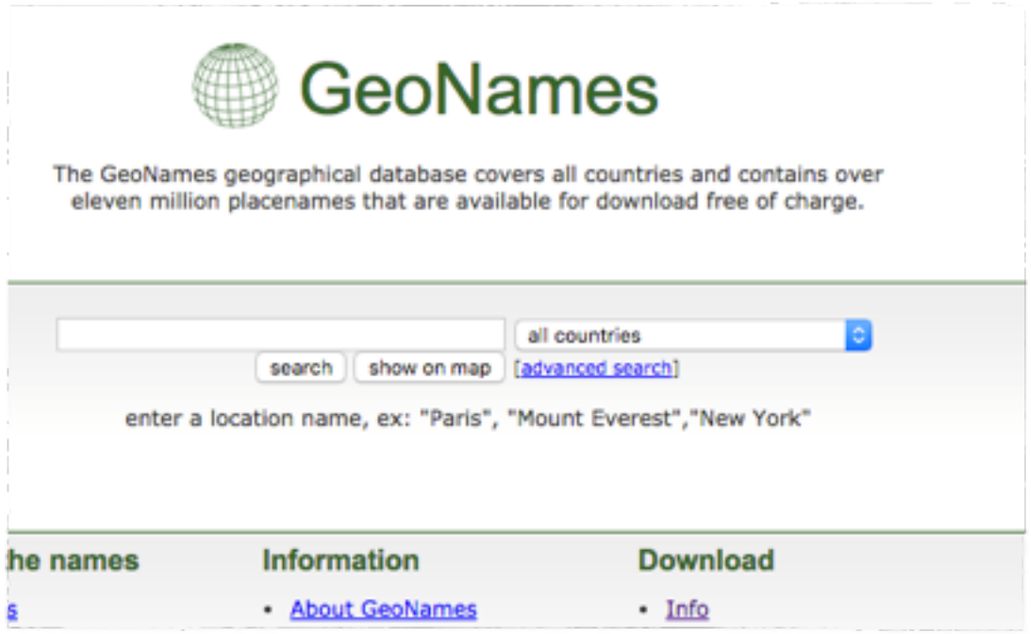
```
import core.data.*;

  public class PeruData1 {
    public static void main(String[] args) {
      DataSource ds = DataSource.connect("https://..
      ds.load();
      ds.printUsageString();
      String[] names = ds.fetchStringArray("NOMBRE")

      System.out.println(names.length);
      System.out.println(names[367]);
    }
}
```

USAGE STRING

```
-----
Data Source: https://commondatastorage.googleapis.com/.../
Ubigeo2010.csv
URL: https://commondatastorage.googleapis.com/.../
Ubigeo2010.csv


The following data is available:
   A list of:
     structures with fields:
     {
       CODDIST : *
       CODDPTO : *
       CODPROV : *
       NOMBRE : *
     }
-----
```

USER-DEFINED CLASS

```
class Geo {
    String name;
    int pop;
    int elev;

    public Geo(String name, int pop, int elev) {
        this.name = name;
        this.pop = pop;
        this.elev = elev;
    }

    public String toString() {
        return String.format("%s (pop. %d): %d m.",
                             name, pop, elev);
    }
}
```



DEMO – ADDITIONAL FEATURES

```
DataSource ds = DataSource.connectAs("TSV",
        "http://download.geonames.org/export/dump/PE.zip");
ds.setOption("fileentry", "PE.txt");
ds.setOption("header",
    "geoid,name,asciiname,altnames,lat,long,feature-class,
     feature-code,cc,cc2,admin1,admin2,admin3,admin4,ppl,
     elev,dem,tz,mod");
ds.load();

Geo g = ds.fetch("Geo", "name", "ppl", "dem");
System.out.println(g);

ArrayList<Geo> places = ds.fetchList("Geo",
                                     "name", "ppl", "dem");
System.out.println(places.size());
for (Geo p : places)
  if (p.name.equals("Arequipa"))
    System.out.println(p);
```

OUTPUT

```
Brazo Tigre (pop. 0): 0 m.
102315
Arequipa (pop. 1218168): 3351 m.
Arequipa (pop. 0): 3164 m.
Arequipa (pop. 841130): 2355 m.
Arequipa (pop. 0): 106 m.
Arequipa (pop. 0): 2327 m.
Arequipa (pop. 0): 404 m.
```

OUTLINE

- Motivation
- Goals
- Usage & Functionality
- Design & Implementation
- Related & Future Work
- Conclusion

MOTIVATION

- The "Age of Big Data"
- Incorporate the use of online data sets in introductory programming courses
  - Provide a simple interface
  - Hide I/O connection, parsing, extracting, data binding

GOALS

- Minimal syntactic overhead
- Direct access via URL (or local file path)
- No requirement of pre-supplied data schemas/templates
- Bind (instantiate) data objects based on user-defined data representations (i.e. student-defined classes)
- Other good stuff
  - Caching
  - Help/usage
  - Error handling/reporting

```
ArrayList<Geo> places
    = ds.fetchList("Geo", ...
```

USAGE

- 3-step approach: • Connect • Load • Fetch
- Infer data format if possible – XML, CSV, JSON
- Display inferred structure of data – printUsageString()
- Fetching atomic values
  - provide a *path* into the data
- Structured data:
  - provide name of class and paths of data to be supplied to the constructor
- Collections: fetchStringArray / fetchArray / fetchList / …

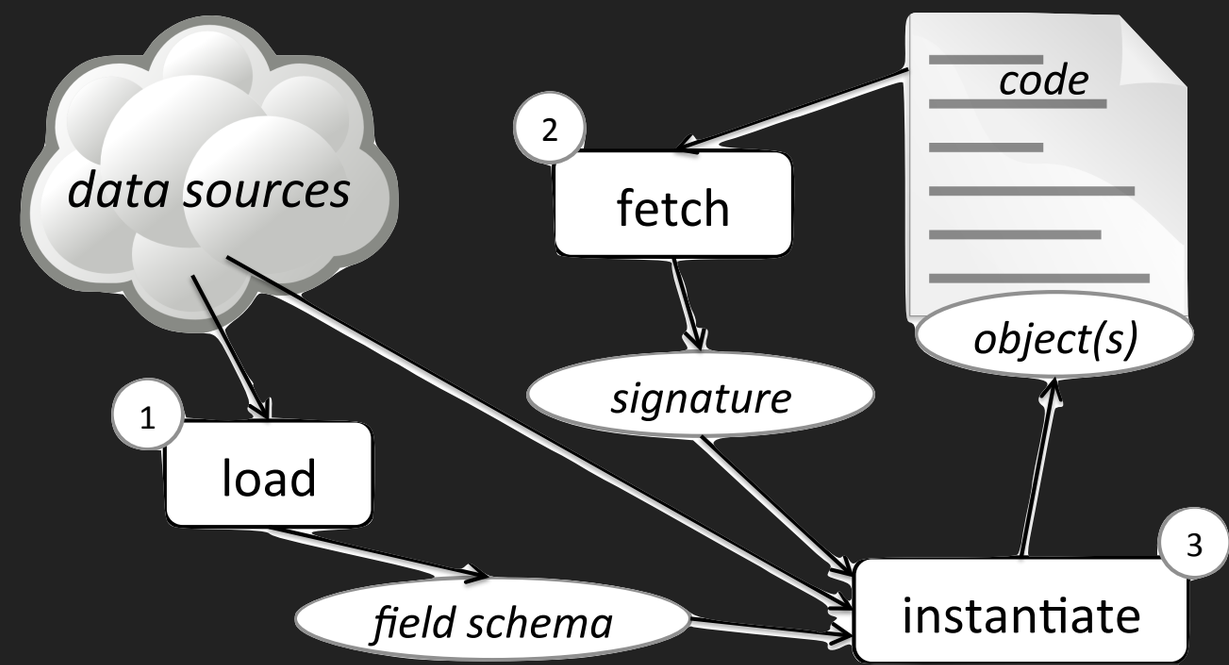
```
ds.fetch("Geo",
         "info/name/std",
         "metrics/pop",
         "phys/elev");
```

OTHER FUNCTIONALITY

- Data source specifications
- Query parameters
- Iterator-based access
- Cache control
- Processing support

DESIGN & IMPLEMENTATION

- Connect
  - prepare URL/path; set parameters, options, data type
- Load:
  - get the data
  - infer a *schema*
- Fetch:
  - build a *signature* for type requested by user
  - *unify* schema with signature - instantiate as objects



EXPERIENCE

- Limited to date: “Creative Computing”
- Tutorial-style labs
- Sample data sets used/discovered by students:

Name (Asterisk indicates data set discovered by students)	Source	Type	Records
*1000 songs to hear before you die	`opendata.socrata.com`	XML	1,000
Abalone data set	UCI Machine Learning Repository	CSV	4,177
*Airport Weather Mashup	NWS + FAA	XML	fixed
*Chicago life expectancy by community	`data.cityofchicago.org`	XML	~80
Earthquake feeds	US Geological Survey	JSON	variable
*Fuel economy data	US EPA	XML	35,430
*Jeopardy! question archive	reddit	JSON	216,930
Live auction data	Ebay	XML	100/page
Magic the Gathering card data	`mtgjson.com`	JSON	variable
Microfinance loan data	Kiva	XML	variable
*SEC Rushing Leaders 2014	ESPN	CSV (manual)	variable

ISSUES

- Finding proper links to raw data (students can have trouble)
- Sites requiring “developer” registration
- Error messages not helpful (yet)
- ~~XML as common intermediate format~~
- Better caching (~~of schemas as well as raw data~~)
  - Streaming, pagination, sampling…

FUTURE

- ~~Redo abstraction layer over data formats~~
- GUI tools
- Multiple language support (Python, Racket)
  - Different language mechanisms to achieve dynamic binding (reflection, macros)
- Additional data formats
  - HTML tables, web scrapers (regexps)
  - Customized for popular APIs (ebay, twitter, etc.)

- **Curriculum resources**
- Evaluation of effectiveness

RELATED WORK & ACKNOWLEDGEMENTS

- CORGIS Dataset Project - http://think.cs.vt.edu/corgis/
- XML Data Access Interfaces
  - JAXB, Castor: schema-based; compile-time setup required
  - FasterXML (Jackson): dynamic binding to POJOs; emphasis on Java → XML direction; tight coupling
- XML schema inference

- Contributions by Steven Benzel, Stephen Jones, Alan Young

CONCLUSION

- Facilitate incorporation of online data sources into programming assignments
  - Painlessly
  - Seamlessly

Use a data set in your next assignment!

cs.berry.edu/sinbad

DATA SOURCE SPECIFICATION FILE

- Data source URL and format.
- Human-friendly name and description, along with URL to a project or informational page about the data source.
- A specification of pre-supplied and user-supplied (required and optional) query parameters or path parameters. The latter are user-provided strings that are substituted in for placeholders in the URL path.
- Programmatic options specific to the particular data source object (such as a header for CSV files).
- Cache settings, such as cache directory path or timeout.
- A data schema defining the exposed data structures and fields from the source with various helpful annotations such as textual descriptions of fields that can be displayed by *printUsageString()*.

DataSource.connectUsing("geospec-pe.spec");

```
{
        "name": "Geographical Data - Peru",
        "format": "TSV",
        "path": "http://download.geonames.org/export/dump/PE.zip",
        "infourl": "http://www.geonames.org/",

        "options": [
    {     "name": "fileentry",
          "value": "PE.txt" },
    {
                "name": "header",
                "value":
"geoid,name,asciiname,altnames,lat,long,feature-class,feature-
code,cc,cc2,admin1,admin2,admin3,admin4,pop,elev,dem,tz,mod"
        }],
}
```

SCHEMAS & SIGNATURES

$$(schema)\ \sigma := * \ \mid\ [_p\sigma] \ \mid\ \{f_{0_{p_0}} : \sigma_0, \ldots\}$$

$$(signature)\ \tau := \tau_B \ \mid\ [\tau] \ \mid\ \mathcal{C}_{\{f_0:\tau_0,\ldots\}}$$

- Primitive, List, or Structure

```
The following data is available:
   A structure with fields:
   {
     row : A list of:
             A structure with fields:
             {
               Address_1 : *
               Electricity_Use_-_Grid_Purchase_kWh : *
               Energy_Cost_ : *
               ...
               Natural_Gas_Use_therms : *
               Property_GFA_-_Self-Reported_ft : *
               Property_Id : *
               Property_Name : *
               ...
               Weather_Normalized_Site_EUI_kBtu-ft : *
               Year_Ending : *
             }
```

```
ds.fetch("Prop",
         "row/Property_Name",
         "row/Year_Ending",
         "row/Energy_Cost_");
```

UNIFICATION

$$(\textit{conversion})\ h := \texttt{parse}B(\delta) \ \mid\ h(\delta[i]) \ \mid\ h(\delta.p) \ \mid\ \texttt{new}\ \mathcal{C}(h_0, \ldots) \ \mid\ \texttt{new}\ \texttt{list}[h_0, \ldots]$$

$\sigma \parallel \tau \Rightarrow h$
means schema σ unifies with signature τ to produce a conversion expression h.

PRIM-PRIM

$$\frac{}{* \parallel \tau_B \Rightarrow \texttt{parse}B(\delta)}$$

PRIM-SINGLETON-COMP

$$\frac{* \parallel \tau \Rightarrow h}{* \parallel \mathcal{C}_{\{f:\tau\}} \Rightarrow \texttt{new}\ \mathcal{C}(h(\delta))}$$

LIST-LIST

$$\frac{\sigma \parallel \tau \Rightarrow h}{[\sigma] \parallel [\tau] \Rightarrow \texttt{new}\ \texttt{list}([h(\delta_0), \ldots])}$$

LIST-STRIP

$$\frac{\sigma \parallel \tau \Rightarrow h}{[\sigma] \parallel \tau \Rightarrow h(\delta_0)}$$

WRAP-LIST

$$\frac{\sigma \parallel \tau \Rightarrow h \qquad \sigma \text{ is not a list schema}}{\sigma \parallel [\tau] \Rightarrow \texttt{new}\ \texttt{list}([h(\delta)])}$$

COMP-STRIP

$$\frac{\sigma \parallel \tau \Rightarrow h}{\{f_p : \sigma\} \parallel \tau \Rightarrow h(\delta.p)}$$

COMP-COMP

$$\frac{\sigma_i \parallel \tau_i \Rightarrow h_i}{\{f_{0_{p_0}} : \sigma_0, \ldots, f_{n_{p_n}} : \sigma_n,\ g_{0_{g_0}} : \sigma_{n+1}, \ldots\} \parallel \mathcal{C}_{\{f_0:\tau_0,\ldots,f_n:\tau_n\}} \Rightarrow \texttt{new}\ \mathcal{C}(h_0(\delta.p_0), \ldots)}$$

BART, ET AL. FIGURE 2

```java
import java.util.List;
import java.util.HashSet;
import realtimeweb.earthquakeservice.main.EarthquakeService;
import realtimeweb.earthquakeservice.domain.Earthquake;

public class EarthquakeDemo {

    public static void main(String[] args) throws EarthquakeException {
        // Use the EarthquakeService library
        EarthquakeService es = EarthquakeService.getInstance();

        es.connect();  // Remove to use the local cache

        // 5 minute delay, but if we use the cache no delay is needed!
        int DELAY = 5 * 60 * 1000;

        HashSet<Earthquake> seenQuakes = new HashSet<Earthquake>();

        // Poll service regularly
        while ( true ) {
            // Get all earthquakes in the past hour
            List<Earthquake> latest = es.getEarthquakes(History.ALL);
            // Check if this is a new earthquake
            for (Earthquake e : latest) {
                if (!seenQuakes.contains(e)) {
                    // Report new earthquakes
                    System.out.println("New quake!");
                    seenQuakes.add(e);
                }
            }
            // Delay to avoid spamming the weather service
            Thread.sleep(DELAY);
        }
    }
}
```

EQUIVALENT

```
import big.data.*;
import java.util.Date;
import java.util.HashSet;
import java.util.List;

public class EarthquakeDemo {
    public static void main(String[] args) {
        int DELAY = 5;   // 5 minute cache delay

        DataSource ds = DataSource.connectJSON("http://earthquake.usgs.gov/earthquakes/fee
        ds.setCacheTimeout(DELAY);

        ds.load();
        ds.printUsageString();

        HashSet<Earthquake> quakes = new HashSet<Earthquake>();

        while (true) {
            ds.load();                          // this only actually reloads data when the
            List<Earthquake> latest = ds.fetchList("Earthquake",
                    "features/properties/title",
                    "features/properties/time",
                    "features/properties/mag",
                    "features/properties/url");
            for (Earthquake e : latest) {
                if (!quakes.contains(e)) {
                    System.out.println("New quake!... " + e.description + " (" + e.date()
                    quakes.add(e);
                }
            }
        }
    }
}
```

PLUS...

```
class Earthquake {                    // this class may be instructor-provided, or left to students to define as an exercise
    String description;
    long timestamp;
    float magnitude;
    String url;

    public Earthquake(String description, long timestamp, float magnitude, String url) {
        this.description = description;
        this.timestamp = timestamp;
        this.magnitude = magnitude;
        this.url = url;
    }

    public Date date() {
        return new Date(timestamp);
    }

    public boolean equals(Object o) {       // introductory CS students would probably implement a simpler version of this
        if (o.getClass() != this.getClass())
            return false;
        Earthquake that = (Earthquake) o;
        return that.description.equals(this.description)
                && that.timestamp == this.timestamp
                && that.magnitude == this.magnitude;
    }

    public int hashCode() {                 // technically, hashCode() should be overridden if equals() is
        return (int) (31 * (31 * this.description.hashCode()
                + this.timestamp) + this.magnitude);
    }
}
```