

**IAT 352 Project 2 - Flickr API + Servlet Code**  
**Alexander James Ehwalt Ryan**  
**301071161**

**Overview:**

The assignment brief required us to have the google map from the previous assignment take a modified version of our first assignment, but instead let us modify the XML to only display areas that were available to be seen on the map, presumably so that, if we had a large XML file, we could display only relevant portions to the user. This required us to take the XML file into the servlet, and based on the position of the user's latitude and longitude as defined in their get request, we would output a similar file(Following the schema set for the first XML file, but with

less entries overall) and dynamic input for photos (We would ignore any photos in our initial XML file, and instead implement our own version that fit the schema for these such that these new photos acted as if they were from the original setup.) would be inserted into the XML file we outputted, to give us dynamic control with Flickr's updated images.

I did not finish the full version of the project, which was to implement the Google Map and have it populate with the markers, as much of my attention was focused on getting the servlet working and running so I could debug it, although this was not the only place I had worked. on code. Some issues that came up for me where the inability to develop in multiple locations (From a debugging perspective), as well as a rather annoying approach to the development environment from the beginning. However, I was able to learn from others who had encountered similar or different problems, and if I were to do this again, I feel I would be better prepared for the assignment.

### **Quirks:**

It is unfortunately worth noting a few quirks of the development environment for this application, as setup and use of tools turned out to be problematic:

- First, I'll note that the development environment assumes an identical folder structure in a E:/ directory, as quirks of my own machine combined with server reading capabilities. The servlet itself had issues finding files in its directory structure to read, and thus in order to read the XML file, I had to, while working on the C:/ directory, use a folder that wasn't my desktop, as my directory used spaces within it, and the workaround of using %20 was not very usable. This can be worked around in the ZoneServlet class with the file locations of the actual XML, XSL, and XSD files.

- Without an appropriate interface for the servlet, it is required that you access it as a GET request, using a link of the format <http://localhost:8084/FlickrAPIProject2/ZoneServlet?latitude=49.188948&longitude=-122.847879&radius=5.0> , where these values are changeable, and the most notable change comes from radius - increasing it to 500 allocates new values from the XML, as more distance is allowed. (See ZoneServlet for more.) Ideally this would be fixed, but complications arose in the index.html/loadContent.js combination, and this was not completed.

- Finally, although photos are queried (See Documentation - ZoneStorage.initatePhotos(...)), it is unfortunate to note that other activities prevented them from being loaded into their appropriate ZonePhoto structure. This also means that photos did not make it into the final product, but the framework is there for them.

### **What went well:**

I am particularly proud of the isolation of the sorting algorithm that the servlet deals with to determine which objects should be used away from the actual implementation of the XML objects - by doing this, each object can control how it is represented, and the structure was

easier to understand.

doGet does the following:

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    if (DEBUG) {
        System.out.println("doGet");
    }
    /*TODO:java.lang.NumberFormatException: For input string: ""49""
sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1224)
java.lang.Double.parseDouble(Double.java:510)
servletPackage.ZoneServlet.doGet(ZoneServlet.java:82)
javax.servlet.http.HttpServlet.service(HttpServlet.java:621)
javax.servlet.http.HttpServlet.service(HttpServlet.java:722)
org.netbeans.modules.web.monitor.server.MonitorFilter.doFilter(MonitorFilter.java:393)

http://localhost:8084/FlickrAPIProject2/ZoneServlet?latitude=49.188948&longitude=-122.847879&radius=5.0
*/
    Double lat = Double.parseDouble(request.getParameter("latitude"));
    Double lng = Double.parseDouble(request.getParameter("longitude"));
    Double radius = Double.parseDouble(request.getParameter("radius")); //8
    String photoSearchTerms = "";
    int numOfPhotoGet = 5;
    System.out.println("Request received");
    org.jdom.Element root = new org.jdom.Element("locationcollection");
    org.jdom.Namespace xsiNS = org.jdom.Namespace.getNamespace("xsi",
        "http://www.w3.org/2001/XMLSchema-instance");

    //TODO: Schema Location?
    root.addNamespaceDeclaration(xsiNS);
    root.setAttribute("noNamespaceSchemaLocation", xsdSrcFilePath, xsiNS); //TODO: Why
to access this file, and how
    for (int i = 0; i < zones.size(); i++) {
        ZoneStorage restnt = zones.get(i); //10
        Double restntLat = Double.parseDouble(restnt.getLatitude());
        Double restntLng = Double.parseDouble(restnt.getLongitude());
        if (DEBUG) {
            System.out.println(getEuclidianDistance(lat, lng, restntLat, restntLng).
                toString() + "radius: " + radius.toString());
        }
        if (getEuclidianDistance(lat, lng, restntLat, restntLng) <= radius) //11
    //TODO:
        restnt.initiatePhotos(restntLat, restntLng, photoSearchTerms,
numOfPhotoGet);
        root.addContent(restnt.convertToElement());
    }
}
org.jdom.Document doc = new org.jdom.Document();
doc.setRootElement(root); //22
```

```

HashMap<String, String> piMap = new HashMap<String, String>();
piMap.put("type", "text/xsl");
piMap.put("href", xslSrcFilePath);
ProcessingInstruction pi = new ProcessingInstruction("xml-stylesheet", piMap);
doc.getContent().add(0, pi);
XMLOutputter xmlOut = new XMLOutputter(org.jdom.output.Format.getPrettyFormat());
response.flushBuffer();
response.getOutputStream().flush();
response.setContentType("text/xml");
// FileOutputStream debug = new FileOutputStream(new File("log.xml")); //TODO: For
debugging?
xmlOut.output(doc, response.getOutputStream()); //TODO: Or will the XML output get
shown if Servlet is directly called?
xmlOut.output(doc, System.out); //TODO: Or will the XML output get shown if Servlet is
directly called?
System.out.println("Done");

System.out.println("Done");
//response.getOutputStream()
}

```

*In the bolded code, we leave the actual implementation of the initiation and rendering to the classes themselves, which not only made it easy to debug, but also allowed me to get around having issues setting up the server - I could work on these in the meantime, allowing these functions:*

```

public Integer initiatePhotos(Double lat, Double lng,
    String photoSearchTerms, int numOfPhotos){
    boolean getGeoLocationCall = true; //12
    try {
        String apiKey = URLEncoder.encode(flickrAPIKey, "UTF8");
        String tags = URLEncoder.encode(photoSearchTerms, "UTF8");
        String accuracy = URLEncoder.encode("16", "UTF8");
        String contentType = URLEncoder.encode("6", "UTF8");
        String hasGeo = URLEncoder.encode("1", "UTF8");
// String latitude = URLEncoder.encode(lat.toString(), "UTF8");
// String longitude = URLEncoder.encode(lng.toString(), "UTF8");
        String radius = URLEncoder.encode("2.5", "UTF8");
        String radiusUnits = URLEncoder.encode("km", "UTF8");
        String extras = URLEncoder.encode("url_t, url_m, geo", "UTF8"); //13
        //TODO: Make private variable
        String flickrPhotoSearchRequestURL = "http://api.flickr.com/services/rest/?"
            + "method flickr.photos.search&api_key" + apiKey
            + "&tags=" + tags + "&accuracy=" + accuracy
            + "&content_type=" + contentType + "&has_geo=" + hasGeo + "&lat=" + latitude
            + "&lon=" + longitude
            + "&radius=" + radius + "&radius_units=" + radiusUnits + "&extras=" + extras + "";
//14
        HttpURLConnection con = (HttpURLConnection) (new
        URL(flickrPhotoSearchRequestURL)).openConnection();
        con.setInstanceFollowRedirects(false); //15
    }
}

```

```

DocumentBuilderFactory fact = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = fact.newDocumentBuilder();
Document doc = builder.parse(con.getInputStream());//16
NodeList list = doc.getElementsByTagName("photo");
int listSize = list.getLength();//17
for (int i = 0; i < listSize; i++) {
    if (i < numOfPhotos) {

        Element element = (Element) list.item(i);
        //18
        //TODO: Pull elements in here, then place into constructor
        ZonePhoto fPhoto = new ZonePhoto();
        fPhoto.setId(element.getAttribute("id"));
        fPhoto.setOwner(element.getAttribute("owner"));
        fPhoto.setSecret(element.getAttribute("secret"));
        fPhoto.setServer(element.getAttribute("server"));
        fPhoto.setFarm(element.getAttribute("farm"));
        fPhoto.setIspublic(element.getAttribute("ispublic"));
        fPhoto.setIsfriend(element.getAttribute("isfriend"));
        fPhoto.setIsfamily(element.getAttribute("isfamily"));
        fPhoto.setUrl_t(element.getAttribute("url_t"));
        fPhoto.setHeight_t(element.getAttribute("height_t"));
        fPhoto.setUrl_m(element.getAttribute("url_m"));
        fPhoto.setHeight_m(element.getAttribute("height_m"));
        fPhoto.setWidth_m(element.getAttribute("width_m"));
        fPhoto.setLatitude(element.getAttribute("latitude"));
        fPhoto.setLongitude(element.getAttribute("longitude"));
        fPhoto.setAccuracy(element.getAttribute("accuracy"));
        fPhoto.setPlace_id(element.getAttribute("place_id"));
        fPhoto.setWoeid(element.getAttribute("woeid"));
        fPhoto.setGeo_is_family(element.getAttribute("geo_is_family"));
        fPhoto.setGeo_is_friend(element.getAttribute("geo_is_friend"));
        fPhoto.setGeo_is_contact(element.getAttribute("geo_is_contact"));
        fPhoto.setGeto_is_public(element.getAttribute("geo_is_public"));*/
        if (getGeoLocationCall) {
            flickrPhotoSearchRequestURL =
                "http://api.flickr.com/services/rest/"
                + "method flickr.photos.geo.getLocation&api_key=" + apiKey
                + "&photo_id=" + element.getAttribute("id");
            con = (URLConnection) (new URL(flickrPhotoSearchRequestURL)
            ).openConnection();
            con.setInstanceFollowRedirects(false);
            fact = DocumentBuilderFactory.newInstance();
            builder = fact.newDocumentBuilder();
            doc = builder.parse(con.getInputStream());
            //19
            NodeList list2 = doc.getElementsByTagName("location");
            if (list2.getLength() > 0) {
                Element element2 = (Element) list2.item(0);
                System.out.println("call1: " + element.getAttribute("latitude") + " " +

```

```

element.getAttribute("longitude"));
        System.out.println("call2: " + element2.getAttribute("latitude") + " " +
element2.getAttribute("longitude"));
        System.out.println("Compare Coordinate from search and
get.geolocation: lat same=" + (element.getAttribute("latitude").
        equals(element2.getAttribute("latitude"))) + "; lng same=" +
(element.getAttribute("longitude").equals(element2.getAttribute("longitude")));
        /*
//        fPhoto.setLatitude(element2.getAttribute("latitude"));
//        fPhoto.setLongitude(element2.getAttribute("longitude"));
    }
    }
//    photos.add(fPhoto);
    } else {
        break;
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
return photos.size();
}

```

- This did the Flickr data, which as you may see, did not get implemented - this still allowed for this to be worked on, however, below, for the XML parsing data:

```

public org.jdom.Element convertToElement() {

    org.jdom.Element restntElement = new org.jdom.Element("location");//20
//    restntElement.setAttribute("id", restnt.getID());
    restntElement.addContent(new org.jdom.Element("id").addContent(this.getIdentifier()));
    restntElement.addContent(new org.jdom.Element("title").addContent(this.getTitle()));
    restntElement.addContent(new org.jdom.Element("lines").addContent(this.getLines()));
    restntElement.addContent(new org.jdom.Element("zone").addContent(this.getZone()));
    restntElement.addContent(new
org.jdom.Element("description").addContent(this.getDescription()));
//    restntElement.addContent(new
org.jdom.Element("detail_link").addContent(restnt.getDetailLink()));
//    restntElement.addContent(new
org.jdom.Element("reviews").addContent(restnt.getReviews()));
//
//    org.jdom.Element rImageElement = new org.jdom.Element("restaurant_image");
//    rImageElement.setAttribute("url", restnt.getRestaurantImageURL());
//    rImageElement.setAttribute("width", restnt.getRestaurantImageWidth());
//    rImageElement.setAttribute("height", restnt.getRestaurantImageHeight());
//    restntElement.addContent(rImageElement);
    restntElement.addContent(new org.jdom.Element("lat").addContent(this.getLatitude()));
    restntElement.addContent(new org.jdom.Element("lon").addContent(this.getLongitude()));
//
//    restntElement.addContent(coordinateElement);
//    org.jdom.Element sPhotoElement = new org.jdom.Element("surrounding_photos");

```

```
//      sPhotoElement.setAttribute("search_tag", restnt.getSurroundingPhotoSearchTag());
      for (int i = 0; i < this.photos.size(); i++) { //21
        ZonePhoto photo = this.photos.get(i);
        //We let each individual photo element render itself - saves time.
        org.jdom.Element photoElement = photo.renderImage();
        //new org.jdom.Element("photo");

        restntElement.addContent(photoElement);
      }
//      restntElement.addContent(sPhotoElement);
      return restntElement;
    }
}
```

### **What could have gone better:**

Quite a lot, actually - this is only the servlet code, although I was able to watch others develop working javascript for their code (Thanks to Amirt and Nathan), I could not implement it fast enough for this project.

Alongside that, I'd honestly recommend to anyone doing this project to not try and get Eclipse to work with Tomcat - NetBeans itself worked quite well for this purpose, and came with a pre-made setup for Tomcat within it's download.

I missed out a lot of functionality, but by working with Valtcho and Andy, I was able to understand the code I was working with significantly better. I would have preferred more time, but this was in part due to the setup time it took to get Tomcat running - if we had started from the beginning of the class, I would have had plenty of time for implementation. This is my own fault, but I feel I learned from this experience and enjoyed what I had completed.

### **Documentation:**

This contains documentation that would have been Javadoc'd into the program time permitted - signatures are in regular font while documentation is in bold.

**Stores all the data related to the "location" xml tag - using the Flickr API, will store the photos for the "location" tag data as appropriately dynamic - also used to convert back into XML with regards to new photo data.**

```
public class ZoneStorage {
```

**Flickr API calling, searching for ones nearby the latitude and longitude provided,  
With appropriatetags as defined in photoSearchTerms, limited to numOfPhotos**

**photos.**

```
public Integer initiatePhotos(Double lat, Double lng,  
    String photoSearchTerms, int numOfPhotos){
```

**This gives us the XML formattable element based off of the data in the class.  
Most notably, it does:**

**<location>**

**<id>identifier</id>**

**...etc.**

**THEN it goes through the photos collected via  
ZonePhoto.java.renderImage(), and stores them like so:**

**<photo>**

**<id>photoIdentifier</id>**

**...etc.**

**</photo>**

**Once all the photos are stored, then it goes and ends the location  
element:**

**</location>**

```
public org.jdom.Element convertToElement() {
```

```
}}
```

**Stores the data of the photos base elements into here. Note: Actual Thumbnail,  
main image to use, and original(Downloadable) data is  
stored lower.**

```
public ZonePhoto{
```

**As in ZoneStorage, renders the XML photo as follows:**

**<photo>**

**<id> identifier</id>**

**...etc.**

**<thumbnail>**

**...see Thumbnail.renderImage()**

**</thumbnail>**

**<main>**

**...See MainImage.renderImage()**

**</main>**

**<download>**

**...See DownloadImage.renderImage()**

**</download>**

**</photo>**

```
public org.jdom.Element renderImage(){
```

```
}}
```

**Stores Thumbnail data. like so:**

```
<thumbnail>
  <t_url>url</t_url>
  <t_width>width</t_width>
  <t_height>height</t_height>

</thumbnail>
```

```
public Thumbnail(String url, String width, String height){
```

**Renders the structure**

```
public org.jdom.Element renderImage() {

}}}
```

**Main Image**

```
<main>
  <m_url>url</m_url>
  <m_width>width</m_width>
  <m_height>height</m_height>

</main>
```

```
public MainPhoto(String url, String width, String height){
```

**Renders the structure**

```
public org.jdom.Element renderImage() {

}}}
```

**Downloadable Image**

```
<download>
  <d_url>url</d_url>
  <d_width>width</d_width>
  <d_height>height</d_height>

</download>
```

```
public class DownloadableImage extends PhotoType {
```

**Renders the structure**

```
public org.jdom.Element renderImage() {

}}}
```

**Deals with the handling of the servlet XML file parsing to create the ZoneStorage class and such, leaving the flickr API calls and the XML rendering to the lower classes to handle. As such, rather basic.**

```
public class ZoneServlet extends HttpServlet {
```

**Performs the actual latitude/longitude regulation, such that the code does not return every object in the XML, only those required. Everything else is delegated to the DataHolder package objects**

```
protected void doGet(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {
```

**Loads XML**

```
private void loadXML(String xmlFilePath) {
```

**Helps determine if the Latitude and Longitude of a element is in radius range - if so, the data is rendered - if not, it is left alone.**

```
private Double getEuclidianDistance(Double x1, Double x2, Double y1, Double  
y2) {  
}}}
```