# CS 10: Problem solving via Object Oriented Programming

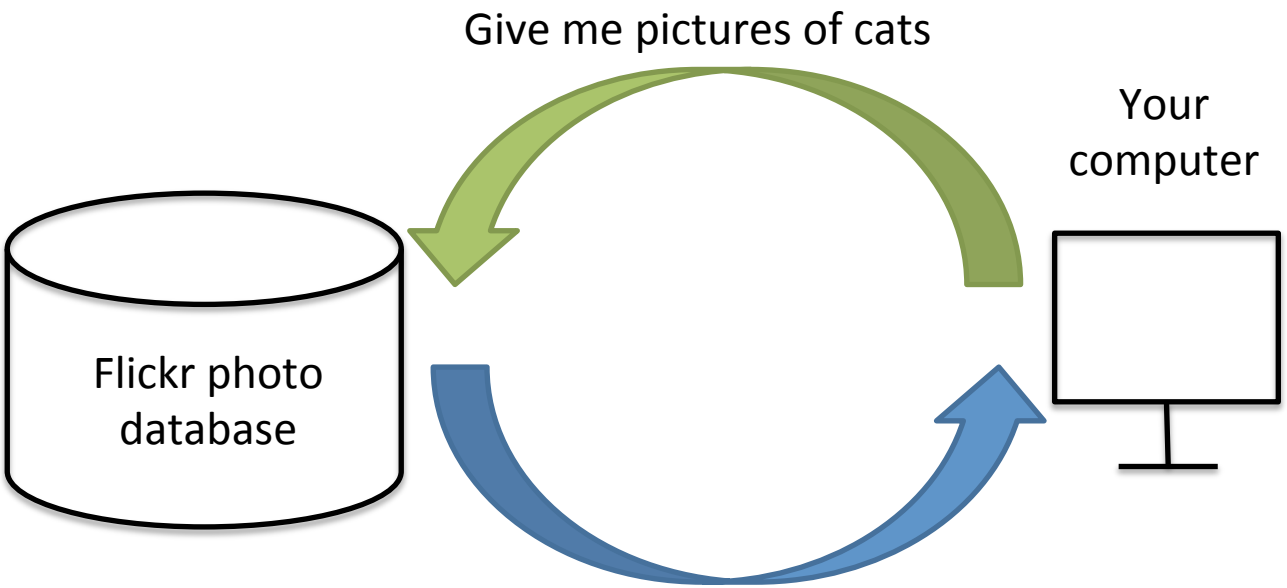
Winter 2017

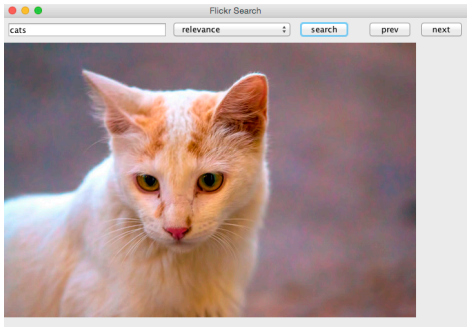
Tim Pierson
260 (255) Sudikoff

## Day 21 – Web Services

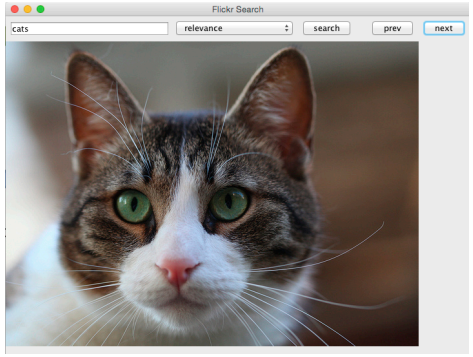
# Big picture: query Flickr and display results

**Overview**




Click next




Click next…

# Agenda

1. Graphical user interface
2. Getting stuff from the web
3. Web services
4. Processing XML
5. Finished product

# Creating Graphical User Interfaces (GUIs) involves graphical elements and listeners

1. Graphical elements are items on the screen the user can interact with
   - Found in Abstract Window Toolkit (AWT) and Swing libraries
   - Provide a wide variety of items such as buttons, text fields, combo boxes
   - Platform (e.g., Windows, Mac) and device independent

2. Listeners respond to user input such as clicking or entering text

# Java graphical elements consists of Containers and Components

**Containers**

JFrame

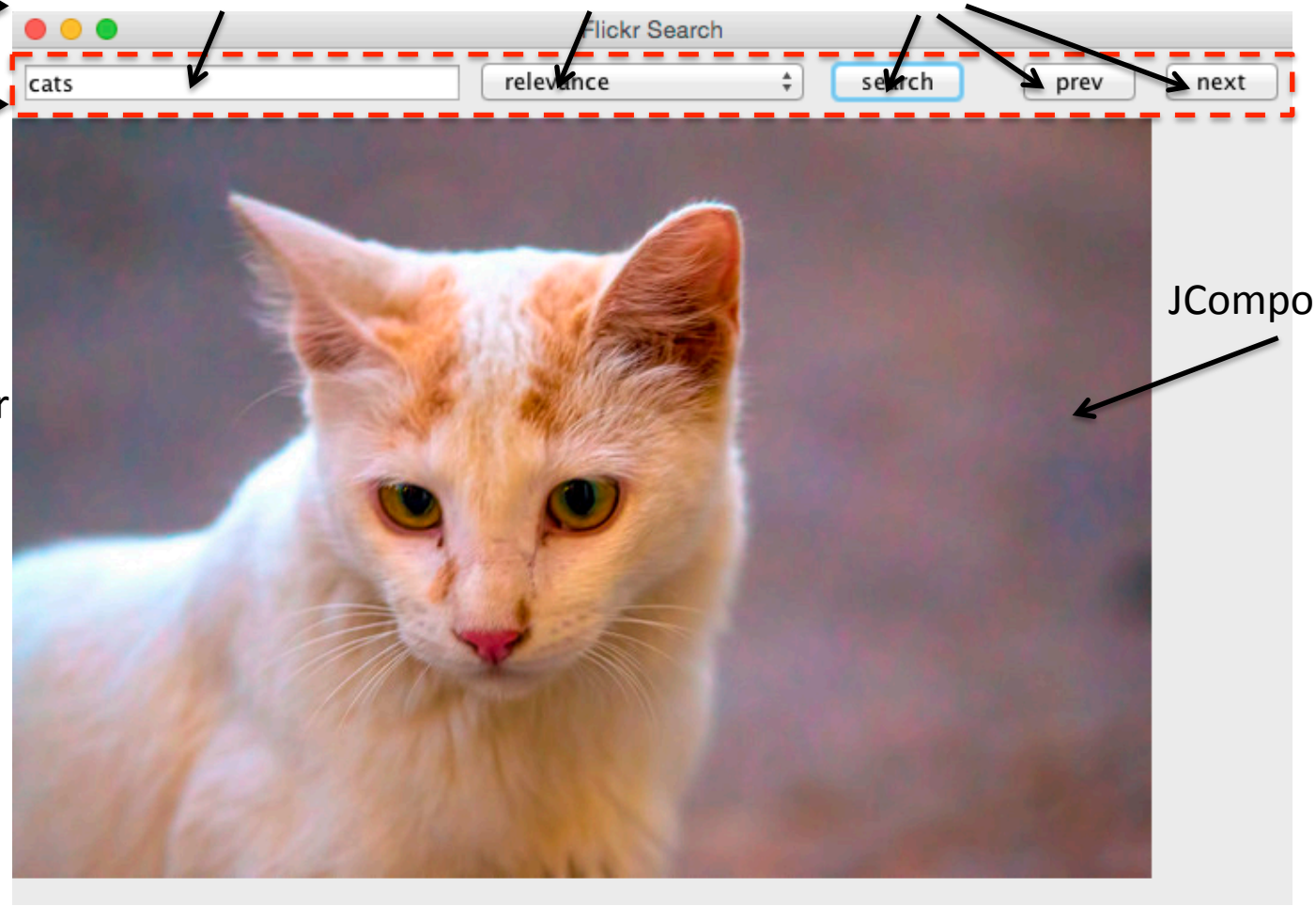
JPanel

**Components**

JTextField

JComboBox

JButton




JComponent

- Containers can hold other containers or components
- May be nested

# Listeners allow us to capture user interaction with graphical elements

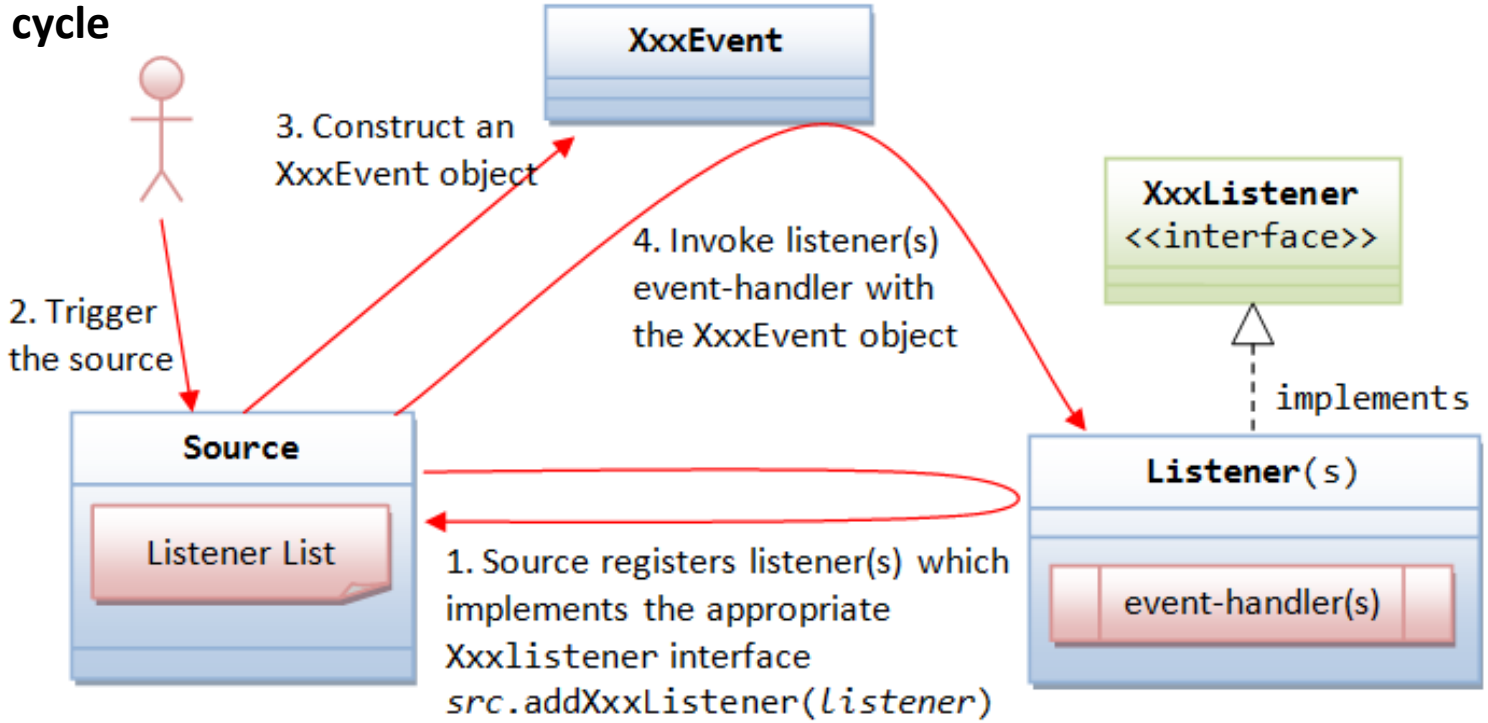
```
// create button control
JButton search = new JButton("search");

//add listener if action taken on button (e.g., clicked)
search.addActionListener(new AbstractAction() {
    public void actionPerformed(ActionEvent e) {
        // this will run if action taken on button
        System.out.println("search button”);
    }
});
```

Listeners are called back when event fires

Located in awt.event.* (import this)

# Events call back listeners

**Event life cycle**




```
// create button control
JButton search = new JButton("search");

//add listener if action taken on button (e.g., clicked)
search.addActionListener(new AbstractAction() {
     public void actionPerformed(ActionEvent e) { //must be implemented to get call back
          // this will run if action taken on button
          System.out.println("search button");
     }
});
```

Image: http://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html

# Creating a Graphical User Interface in Java is tedious without a GUI development tool

**FlickrSearchCore.java**

- Run to show what we are trying to accomplish – windows with a few buttons, text entry, and drop down box (otherwise window is blank, photos from Flickr will go in main window portion)
- Moving away from `DrawingGUI`, putting GUI development in this file
- You can hand code GUI layouts, but *far* easier to use a GUI design tool, here we do it by hand
- FlickrSearchCore extends `JFrame`, Java's graphical window class
- Constructor creates a new canvas of type `JComponent`, point out the use of anonymous class inside new `JComponent`
- `ContentPane` is the main container, `canvas` holds the pictures from Flickr in the container, `gui` holds buttons, etc at the top
- Most setup occurs in `setupGUI` method

# Creating a Graphical User Interface in Java is tedious without a GUI development tool

**FlickrSearchCore.java – setupGUI method**

- Creates button called `prevB` and adds a listener for button events
- Same thing for `nextB`
- Creates `JComboBox` with `sorts` options with listener for events to allow user to specify how to sort images (relevance, date, etc)
- Creates `JTextField` to allow user to input search criteria
- Adds search button with listener
- Package above components into a `Panel`

- Finally, add `canvas` and `gui` to `ContentPane` for window

# Agenda

1. Graphical user interface
2. Getting stuff from the web
3. Web services
4. Processing XML
5. Finished product

# To transfer data between computers we use pre-defined protocols

**Network protocols**

- Protocols define up front how data will be exchanged so everyone knows the "rules"
- There are dozens of protocols used for different purposes:
  - TCP/IP, FTP
  - Wi-Fi, Bluetooth
- HyperText Transfer Protocol (HTTP) is the protocol commonly used by the World Wide Web to get HyperText Markup Language (HTML) documents that describes how to render a web page
- We use a Uniform Resource Location (URL) to specify what page we want to get:

http://www.cs.dartmouth.edu/~tjp/cs10/index.php

Protocol
how we will talk

Computer
that has data

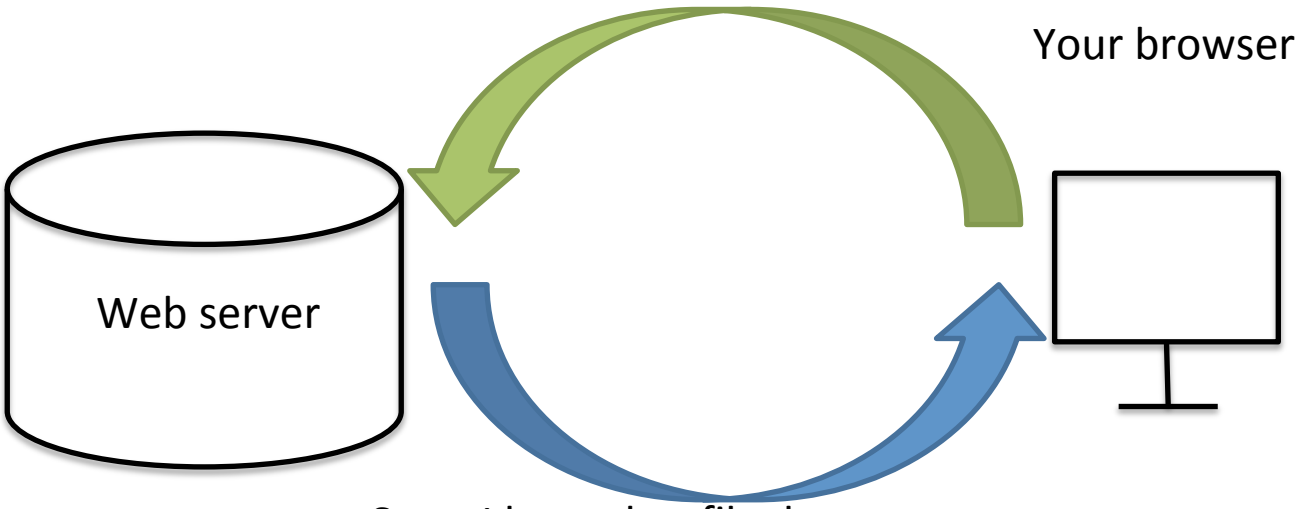
Directory where
data located

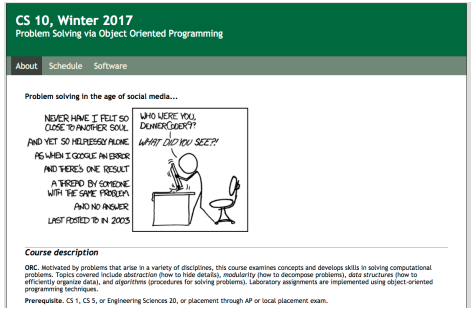
File (assume index.html or
index.php if not provided)

# Client makes a request for a resource to a Server; Server responds to request

**Process**

Give me this file:
http://www.cs.dartmouth.edu/~tjp/cs10/index.php




Browser interprets HTML data and displays page

Sure, I have that file, here you go:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8" />
<title>CS 10 | Problem solving | Winter 2017</title>
</head>

<body>
<div id="page">
<div id="header">
  <div id="title">CS 10, Winter 2017</div>
  <div id="subtitle">Problem Solving via Object Oriented Programming</div>
</div> …
```

# Java makes it easy to get HyperText Markup Language (HTML) from the web

**Getting HTML from the web**

```
public class WWWGetTry {
      public static void main(String[] args) {
            try {
                  // Create the URL; can throw MalformedURL
                  URL url = new URL("http://www.cs.dartmouth.edu/~tjp/cs10/index.php");
                  System.out.println("*** getting " + url);

                  // Create the reader for the stream; can throw IO
                  BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));

                  // Read the lines; can throw IO
                  try {
                        String line;
                        while ((line = in.readLine()) != null) {
                              System.out.println(line);
                        }
                  }
                  // Be sure to close the reader, whether or not reading was successful
                  finally {
                        in.close();
                  }
            }
            catch (MalformedURLException e) {
                  System.err.println("bad URL");
            }
            catch (IOException e) {
                  System.err.println("problem opening/reading/closing");
            }

            System.out.println("*** done");
      }
}
```

# Agenda

1. Graphical user interface
2. Getting stuff from the web
3. Web services
4. Processing XML
5. Finished product

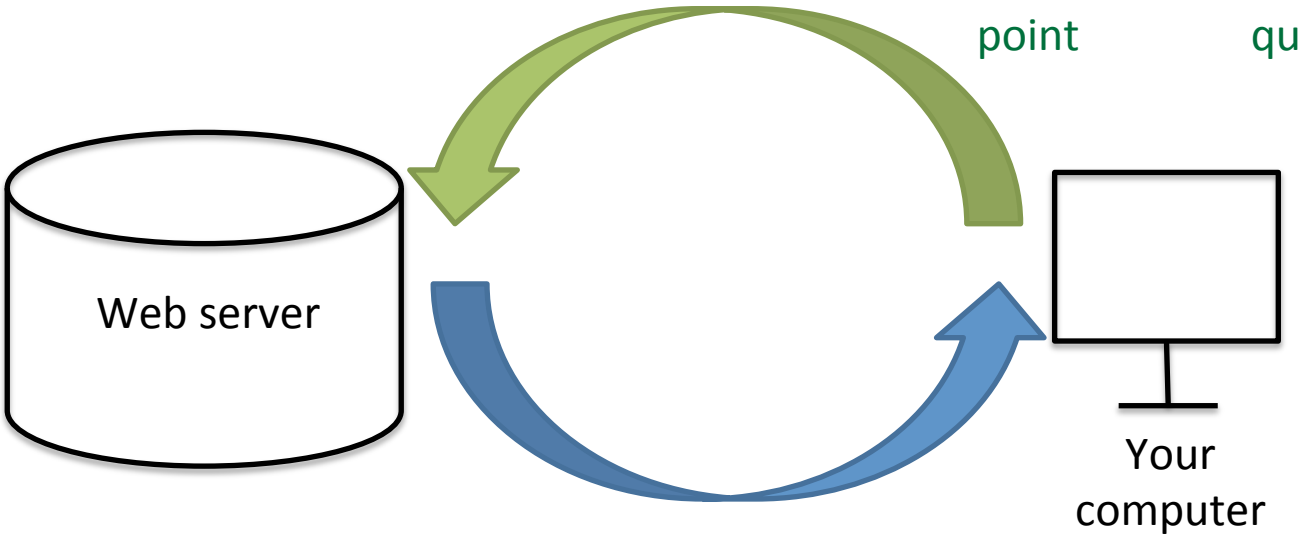
# We can use web services to get data (as opposed to HTML) from a server

**Web service process**

Give me this data:

http://www.cs.dartmouth.edu/~tjp/cs10/hello.php?name=tim&color=blue

Location | Web service end point | Parameters in query string




REST (Representational State Transfer) uses HTTP to transfer data

Sure, I have that data, here you go:

**XML** **OR** **JSON**

```
<Person>
  <name>Tim</name>
  <class>2021</class>
</Person>
```

```
{"Person":
  {"name":"Tim",
   "class":2021}
}
```

# REST web service example

**Enter the following addresses in web browser**

- http://cs.dartmouth.edu/~tjp/cs10/hello.php?name=tim
- http://cs.dartmouth.edu/~tjp/cs10/hello.php?name=tim&color=blue

```
<?php
	$name = $_GET['name'];
	$color = $_GET['color'];
	if (isset($color)) {
		echo 'Hello there '.$name.', thanks for stopping by. My favorite color is '.$color. ' too! ';
	}
	else {
		echo 'Hello there '.$name.', thanks for stopping by!';
	}
?>
```

# Agenda

1. Graphical user interface
2. Getting stuff from the web
3. Web services
4. Processing XML
5. Finished product

# eXtensible Markup Language (XML) is a popular way to representing data

**Sample XML for course enrollment**

Start of enrollment tag

```
<enrollment>
    <course department="CS" number="1" term="17W">
        <student name="Alice" year="20" />
        <student name="Bob" year="19" />
        <student name="Charlie" year="18" />
    </course>
    <course department="CS" number="10" term="17W">
        <student name="Delilah" year="19" />
        <student name="Elvis" year="00" />
        <student name="Flora" year="20" />
    </course>
</enrollment>
```

End of enrollment tag

**XML**

- XML groups data with an opening and closing tag

# eXtensible Markup Language (XML) is a popular way to representing data

**Sample XML for course enrollment**

Start of enrollment tag

```
<enrollment>
    <course department="CS" number="1" term="17W">
        <student name="Alice" year="20" />
        <student name="Bob" year="19" />
        <student name="Charlie" year="18" />
    </course>
    <course department="CS" number="10" term="17W">
        <student name="Delilah" year="19" />
        <student name="Elvis" year="00" />
        <student name="Flora" year="20" />
    </course>
</enrollment>
```

Nested tag called "course"

Another nested tag called "course"

End of enrollment tag

**XML**

- XML groups data with an opening and closing tag
- Tags can be nested

# eXtensible Markup Language (XML) is a popular way to representing data

**Sample XML for course enrollment**

Course tag attributes: department = “CS”, number =1, term=“17W”

```
<enrollment>
    <course department="CS" number="1" term=”17W">
        <student name="Alice" year="20" />
        <student name="Bob" year="19" />
        <student name="Charlie" year="18" />
    </course>
    <course department="CS" number="10" term=”17W">
        <student name="Delilah" year="19" />
        <student name="Elvis" year="00" />
        <student name="Flora" year="20" />
    </course>
</enrollment>
```

Student tags attributes: name=“Flora”, year=“20”

**XML**

- XML groups data with an opening and closing tag
- Tags can be nested
- Tags can have attributes

# eXtensible Markup Language (XML) is a popular way to representing data

**Sample XML for course enrollment**

```
<enrollment>
    <course department="CS" number="1" term="17W">
        <student name="Alice" year="20" />
        <student name="Bob" year="19" />
        <student name="Charlie" year="18" />
    </course>
    <course department="CS" number="10" term="17W">
        <student name="Delilah" year="19" />
        <student name="Elvis" year="00" />
        <student name="Flora" year="20" />
    </course>
</enrollment>
```

**XML**

- XML groups data with an opening and closing tag
- Tags can be nested
- Tags can have attributes
- Typically web services provide documentation to help you interpret the attributes

# Flickr uses XML to return information about photos it stores

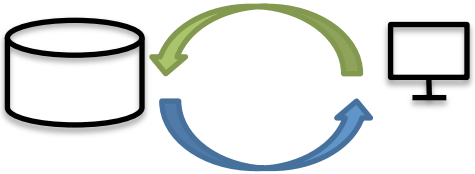
**Simplified Flickr XML data from search**

**Querying Flickr for "dartmouth"**

https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=KEYHERE&text=dartmouth*&sort=relevance&per_page=10*

# Flickr uses XML to return information about photos it stores

**Simplified Flickr XML data from search**



**Querying Flickr for "dartmouth"**

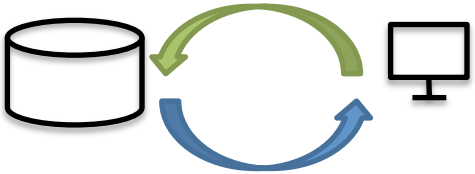
https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=KEYHERE&text=dartmouth*&sort=relevance&per_page=10*

**Returns XML with information about photos of Dartmouth**

```
<rsp stat="ok">
    <photos page="1" pages="1111" perpage="10" total="11106">
        <photo id="3839269905" secret="5513273158"  server="3245" farm="4" />
        <photo id="3840057696" secret="c9428b8fb3" server="3434" farm="4" />
        .
        .
        .
    </photos>
</rsp>
```

# Flickr uses XML to return information about photos it stores

**Simplified Flickr XML data from search**



**Querying Flickr for "dartmouth"**

https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=KEYHERE&text=dartmouth*&sort=relevance&per_page=10*

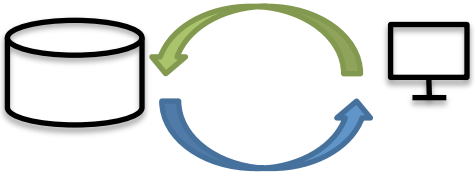
**Returns XML with information about photos of Dartmouth**

```
<rsp stat="ok">
    <photos page="1" pages="1111" perpage="10" total="11106">
        <photo id="3839269905" secret="5513273158"  server="3245" farm="4" />
        <photo id="3840057696" secret="c9428b8fb3" server="3434" farm="4" />
        .
        .
        .
    </photos>
</rsp>
```

Response status is ok
Photos grouped into photos tag
Each photo in its own tag with information describing photo and where to find it

# Flickr uses XML to return information about photos it stores

**Simplified Flickr XML data from search**



**Querying Flickr for "dartmouth"**

https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=KEYHERE&text=dartmouth*&sort=relevance&per_page=10*

**Returns XML with information about photos of Dartmouth**

```
<rsp stat="ok">
    <photos page="1" pages="1111" perpage="10" total="11106">
        <photo id="3839269905" secret="5513273158"  server="3245" farm="4" />
        <photo id="3840057696" secret="c9428b8fb3" server="3434" farm="4" />
        .
        .
        .
    </photos>
</rsp>
```

**Flickr documentation says that photos can be retrieved with:**
http://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}.jpg

http://farm4.staticflickr.com/3245/3839269905_5513273158.jpg

# Agenda

1. Graphical user interface
2. Getting stuff from the web
3. Web services
4. Processing XML
5. Finished product

# Finished product is in FlickrSearch.java

**FlickrSearch.java**

- Run to show what we are trying to accomplish
- Get Flickr key from course web page – don't abuse it!
- Most of the action is in `loadImages`
  - Build Flickr query
  - Use `BufferedReader` to read XML data from Flickr server
  - Use Java's XML parser to handle data
  - Loop over all photos
    - Read photo in `BufferedImage`
    - Store in `images` instance variable
  - Search button reads from text box and queries Flickr
  - Repaint causes `canvas` to display `cur` image