CS 10: Problem solving via Object Oriented Programming

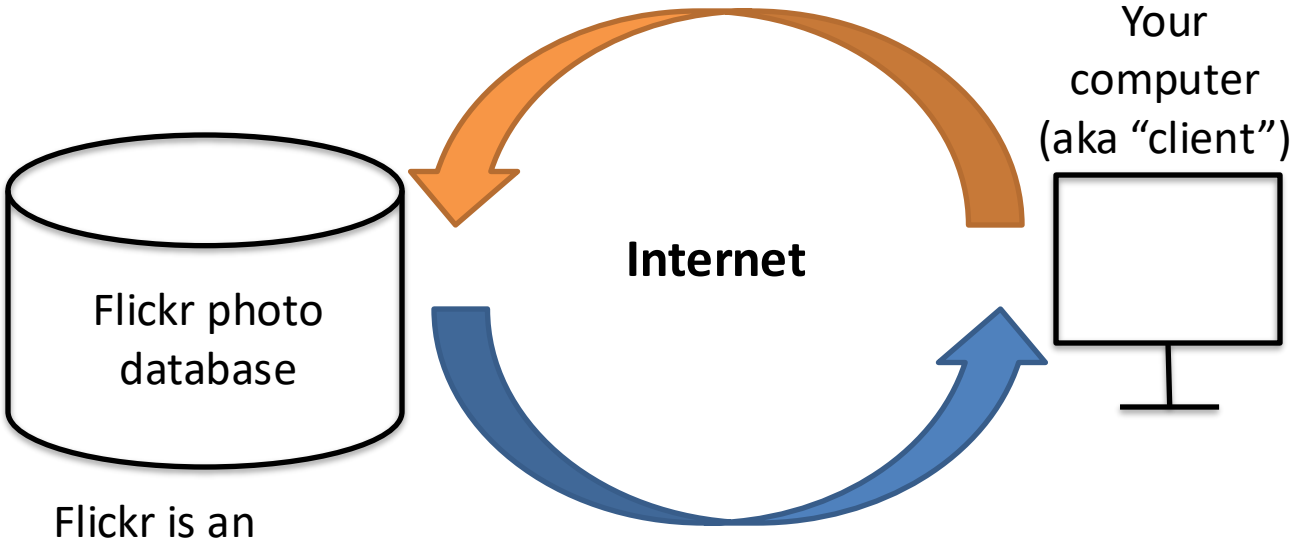
Web Services

DEMO: FlickrSearchJSON.java

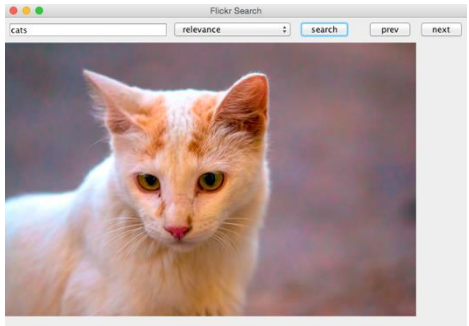
1. Create GUI for displaying photos
2. Ask Flickr for photos matching search term
3. Get back a data stream with information about photos in Flickr database matching search term
4. Fetch each photo described in data stream and display them one at a time

Big picture: query online photo database Flickr and display results

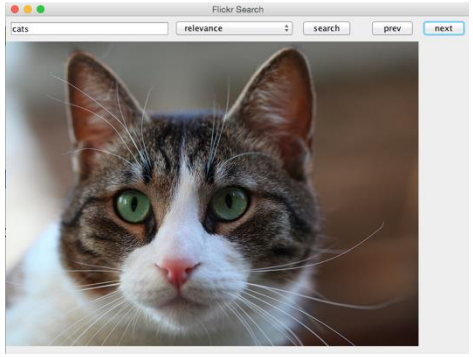
Overview



Flickr is an online photo database



Click next



Click next…

Questions:

1. **How do we create a GUI to display images?**
2. **How can we get data from the web?**
3. **How do we use the data once we get it?**

Agenda

1. Creating Graphical User Interfaces (GUIs)
2. Getting data from the web
3. Web services
4. Processing data
5. Finished product

Today we use JSON

Some notes about older XML format at end of slides

Creating Graphical User Interfaces (GUIs) adds graphical elements and listeners

Two step process to create GUIs

Graphical elements include:

- Buttons
- Text fields
- Combo boxes
- Containers that hold other elements

We tell Java what graphical elements to put on the screen and where to place them

Event listeners call back our code when a user interacts with a graphical element

Listeners get detailed information about the interaction (e.g., which key was pressed, which button is clicked)

In practice, these two steps are often done by different teams

Java graphical elements consists of Containers and Components

Containers

JFrame

JPanel

Components

JTextField

JComboBox

JButton

JComponent

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

Adding these elements manually is tedious
Graphic design tools make life easier
Today we do it the old fashioned way

Step 1: Add graphical elements

FlickrSearchCore.java

```
public class FlickrSearchCore extends JFrame {
    private static final int imageWidth = 640, imageHeight = 640;    // medium 640 on flickr

    private JComponent canvas;                                       // drawing component
    private JTextField queryF;                                       // GUI text field for query
    private String sort = "relevance";                               // how to sort when search

    public FlickrSearchCore() {
        super("Flickr Search");

        // Create our graphics-handling component, sized to hold the images
        canvas = new JComponent() {
            public void paintComponent(Graphics g) {
                super.paintComponent(g);
                // will add code here to draw the current image
            }
        };
        canvas.setPreferredSize(new Dimension(imageWidth, imageHeight));

        // Create the GUI components
        JPanel gui = setupGUI();

        // Put the GUI and the canvas in the panel, one at the top and one taking the rest of
        Container cp = getContentPane();
        cp.setLayout(new BorderLayout());
        cp.add(gui, BorderLayout.NORTH);
        cp.add(canvas, BorderLayout.CENTER);

        // Boilerplate
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
        setVisible(true);
    }
```

Extends JFrame

Create JComponent container hold images

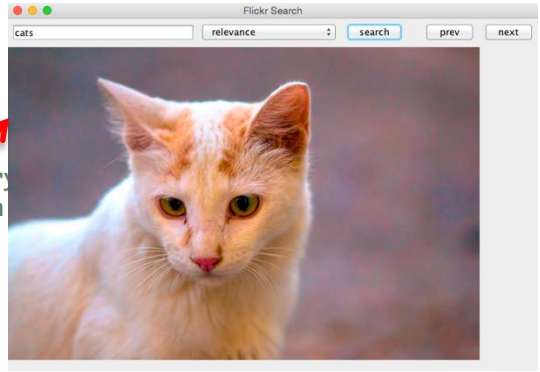
Set JComponent size

Will add code here to display images in JComponent

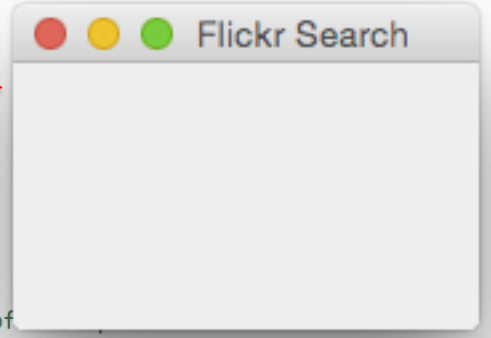
Create a panel to hold buttons (*setupGUI()* is our method on next slide)

Common boilerplate

Add JPanel (button panel) and JComponent (images) to Content Pane



JFrame Content Pane holds Containers



JFrame creates a blank window with a title, same as DrawingGUI

Set up JPanel that holds buttons, text and drop down box

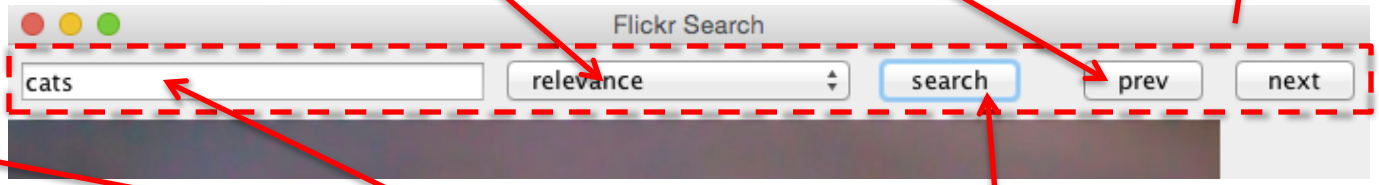
FlickrSearchCore.java

```java
private JPanel setupGUI() {
    // prev button steps backward through images
    JButton prevB = new JButton("prev");
    prevB.addActionListener(new AbstractAction() {
        public void actionPerformed(ActionEvent e) {
            // will add code here to move to the previous image
            System.out.println("prev");
        }
    });

    // sort dropdown (combobox) lists possible ways to sort
    String[] sortOrders = { "relevance", "interestingness-desc", "interestingness-asc",
            "date-taken-desc", "date-taken-asc" };
    JComboBox sortB = new JComboBox(sortOrders);
    sortB.addActionListener(new AbstractAction() {
        public void actionPerformed(ActionEvent e) {
            sort = (String)((JComboBox)e.getSource()).getSelectedItem();
            System.out.println(sort);
        }
    });

    // text field for the search query
    queryF = new JTextField(20);

    // search button fires off the search
    JButton search = new JButton("search");
    search.addActionListener(new AbstractAction() {
        public void actionPerformed(ActionEvent e) {
            // will add code here to fire off the search
            System.out.println("search for '" + queryF.getText() + "' by " + sort);
        }
    });

    // Put all the components in a panel
    JPanel gui = new JPanel();
    gui.setLayout(new FlowLayout());
    gui.add(queryF);
    gui.add(sortB);
    gui.add(search);
    gui.add(new JSeparator(SwingConstants.VERTICAL));
    gui.add(prevB);

    return gui;
}
```

Creates JPanel to hold buttons and dropdown control

Create "prev" Jbutton and listener that fires when button clicked

Create drop down list



Create JTextField
Will search for photos with keywords entered here

Finally add all elements to JPanel and return

Create "search" JButton and listener
Will search for photos with keywords from JTextField

Step 2: Add event listeners that wait for events on graphical elements

Create "search" JButton graphical element

```
// 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");
        }
});
```

Add a listener that will fire when the button is clicked

Here just print that button was clicked

This declaration is called an anonymous class – never gets a name, but has access to instance variables

ActionEvent is an Object that gives details about the event that just occurred (e.g., button click)

Listeners are called back when event fires
Located in awt.event.* (import this)

Agenda

1. Creating Graphical User Interfaces (GUIs)
2. Getting data from the web
3. Web services
4. Processing data
5. Finished product

To transfer data between computers we use pre-defined protocols

Network protocols

- Network protocols define 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 describe how to render a web page
- We use a Uniform Resource Locator (URL) to specify what page we want to get:

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

Protocol: how we will talk (http)

Computer that has data

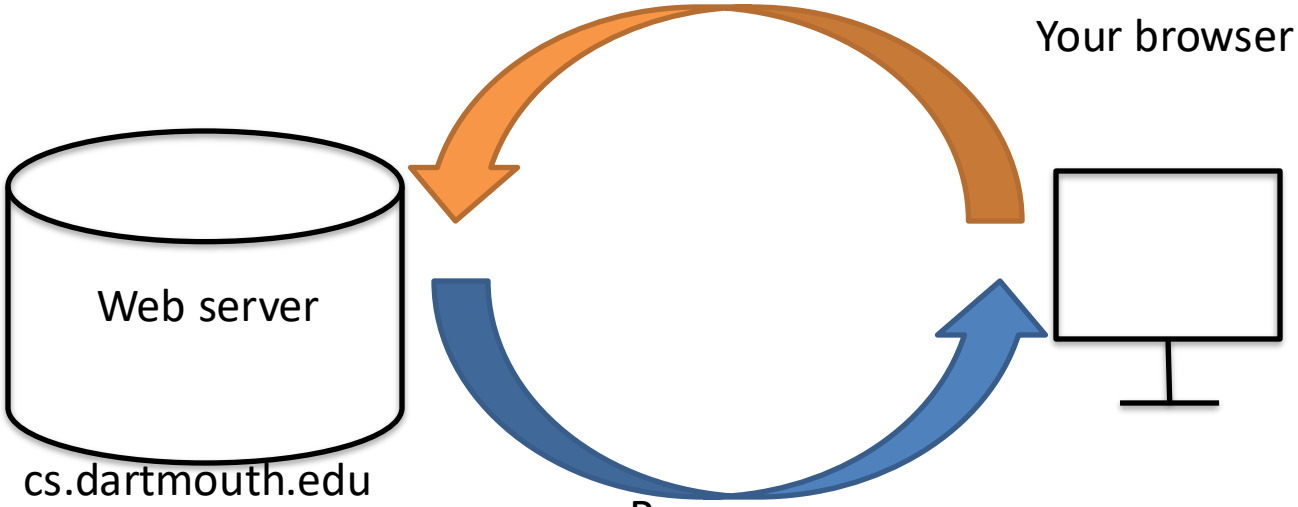
Directory where data located

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

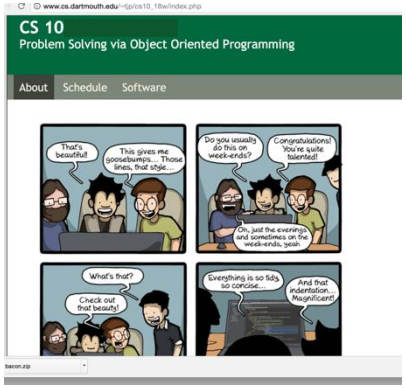
Client makes a request to a Server for a web page; Server responds to request

Process

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



Response:



Browser interprets HTML text and renders page

Big idea:

- **Client makes request to server for web page**
- **Server responds to client by sending text file**

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8" />
<title>CS 10 | Problem solving </title>
<link rel="stylesheet" type="text/css" media="screen" href="cs10.css" />
</head>

<body>
<div id="page">
<div id="header">
 <div id="title">CS 10: Problem Solving via Object Oriented Programming</div>
```

A web page is simply a text document with a description of what to display on the screen (and maybe some Javascript for user interaction) in a format called HTML

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

WWWGet.java

Tell Java where to look for HTML document Location called URL – Uniform Resource Location

URL:

- **Protocol – https (secure version of http)**
- **Server – cs.dartmouth.edu**
- **Location– /~tjp/cs10/notes21.html**

Create BufferedReader to read from URL like reading from file

```
public class WWWGet {
    public static void main(String[] args) throws Exception {
        // Open a stream reader for processing the response from the URL
        URL url = new URL("https://www.cs.dartmouth.edu/~tjp/cs10/ notes20.html");
        System.out.println("*** getting " + url);
        BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));
        // Read lines from the stream, just like reading a file
        String line;
        while ((line = in.readLine()) != null) {
            System.out.println(line);
        }
        in.close();
        System.out.println("*** done");
    }
}
```

Read HTML line by line

Close reader in finally block

Big idea:

- **Java abstracts a lot of messy details for connecting over HTTP so we don't have to deal with it (take CS 60 for more details)**
- **Java lets us read data over the web like we read a file on our local computer**

DEMO: WWWGet.java

Read data from CS web server

Get HTML at:
https://www.cs.dartmouth.edu/~tjp/cs10/notes20.html

Write HTML to console line by line

Sample code WWWGetTry.java does the same, but has more error checking

Agenda

1. Creating Graphical User Interfaces (GUIs)
2. Getting data from the web
3. Web services
4. Processing data
5. Finished product

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

Web service process

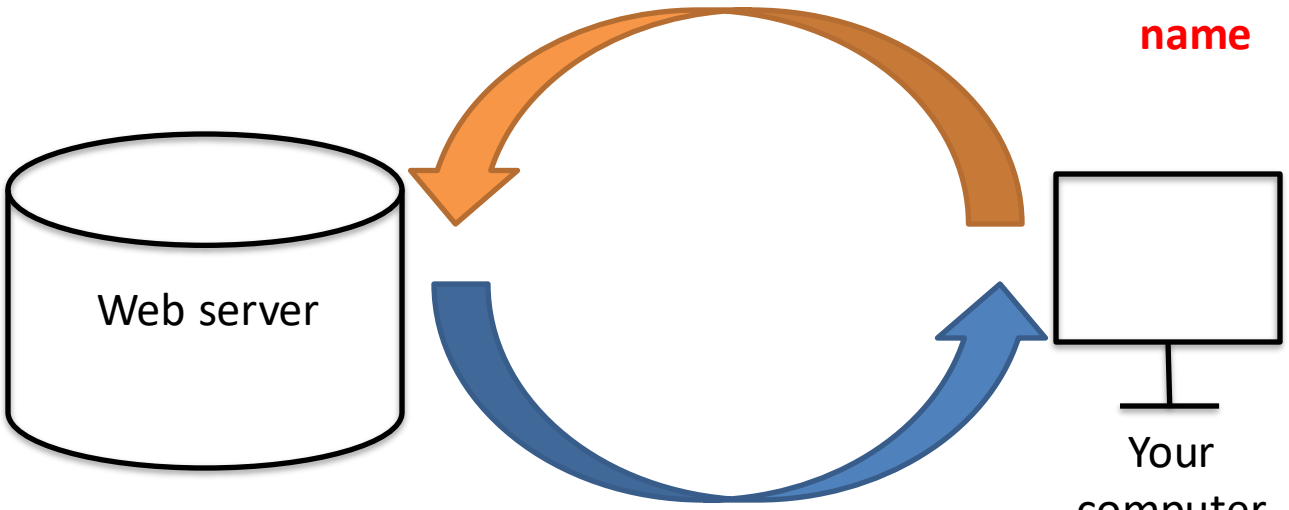
I'd like to get data on student with id=123:

http://www.cs.dartmouth.edu/~tjp/cs10/student.php?id=123

Protocol and location

Web service name

Parameters in query string provided to server



Web service responds with data corresponding to query string parameters

REST (Representational State Transfer) uses HTTP to transfer data

Big idea:

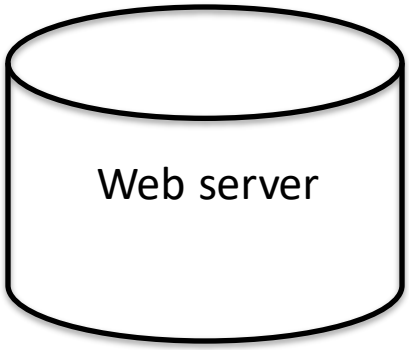
- **Client makes request to server for *data***
- **Server responds to client's request**
- **Intent of web service is for a *program* rather than a human (or a browser) to get data**

Server-side REST web service can return data that does not have to be HTML

Enter the following addresses in web browser

http://cs.dartmouth.edu/~tjp/cs10/student.php?id=123

Query string begins after "?"
Format: param=value
Can have more than one parameter, separate them by &



Request causes student.php code to run on the "server side"

- Reads parameter id=123 from query string
- Looks up data on student with id=123
- Returns information about student with that id

Student Information
Name: Alice
ID: 123
Major: CS
Grades:
CS1: A
CS10: A
CS50: A-

- **Student information returned to client**
- **Information is not HTML, just text**
- **Would prefer a consistent format for data returned**

Agenda

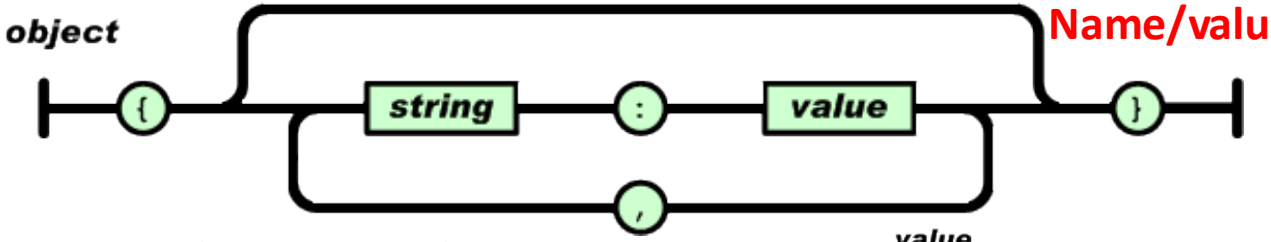
1. Creating Graphical User Interfaces (GUIs)
2. Getting data from the web
3. Web services
4. Processing data
5. Finished product

JSON is a popular way for web services to format data when responding to requests

JSON (JavaScript Object Notation) has two high-level structures

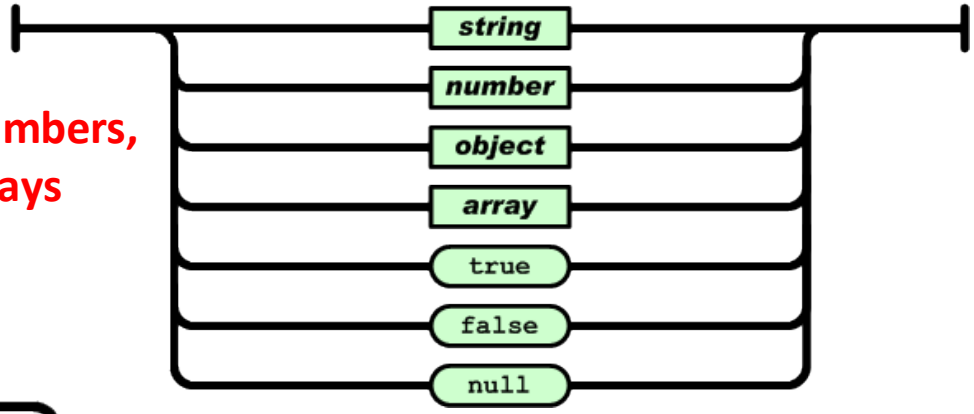
1. **Objects: collection of name/value pairs**

Objects are unordered name/value pairs
Begin with { and end with }
Name/value pairs separated by commas

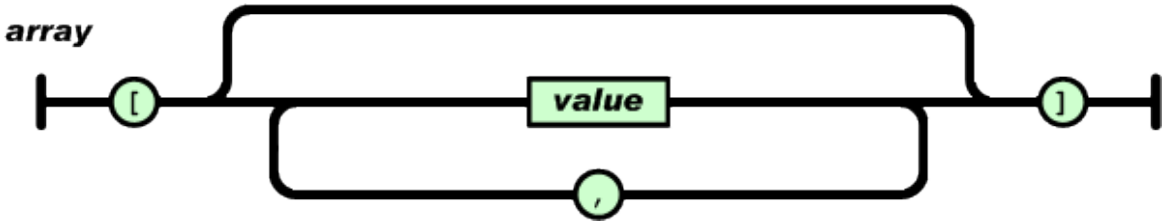


Does this format look familiar to other structures we've seen in this class?
Finite Automata

- Values can be strings, numbers, booleans, objects, or arrays
- Very powerful "nesting"



2. **Arrays: ordered list of values**



Arrays are ordered
Begin with [and end with]
Items separated by commas

Source: www.json.org

JSON provides a consistent way to send data between clients and servers

JSON version of student.php data

```
{
    "Student" : {
        "Name" : "Alice",
        "ID" : 123,
        "Major" : "CS",
        "Grades" : [
            {"CS1" : "A"},
            {"CS10" : "A"},
            {"CS50" : "A-"}
        ]
    }
}
```

Start with JSON object to hold data

Declare name/value as Student with an object as value

Name/value pairs separated by commas
ID has numeric value

- **Name/value pair for Grades, where value is an array of objects**
- **Array has one object for each course this student has taken**
- **Objects are course name/grade received**

Web services that return data provide documentation describing how the data is formatted – read the docs!

JSON formatted data returned is simply text document, must parse it to convert to Java ADTs we know and love

Java parses JSON text into familiar data structures

JSON	Java
String	String
Number	Number
True/false	Boolean
Null	Null
Array	List (JSONArray)
Object	Map (JSONObject subclass of HashMap)

Student object is a Java Map
We can retrieve items with *get()*
***student.get("Name")* returns "Alice"**

```
{
    "Student" : {
        "Name" : "Alice",
        "ID" : 123,
        "Major" : "CS",
        "Grades" : [
            {"CS1" : "A"},
            {"CS10" : "A"},
            {"CS50" : "A-"}
        ]
    }
}
```

Grades is a List
We can retrieve Grades with *student.get("Grades")*
We can loop through the the array items using an iterator or a standard "for" loop

NOTE: Follow instructions on course web page to install JSON parser

Flickr can use JSON to return information about photos it stores

Simplified Flickr JSON data from search

Querying Flickr for "dartmouth"

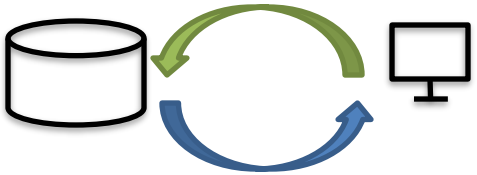
URL (protocol, server, location)

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

Query string:
method = search photos (flickr.photos.search)
api_key = find on Canvas under Pages (identifies us to Flickr, don't abuse key!)
text = find photos matching this text (dartmouth)
sort = by relevance, by date, etc (relevance)
per_page = how many photos to return (10)
format = return data in this format (json)

Flickr can use JSON to return information about photos it stores

Simplified Flickr JSON data from search



Querying Flickr for "dartmouth"

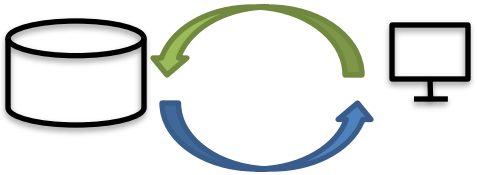
https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=KEYHERE&text=dartmouth&sort=relevance&per_page=10&format=json

Returns JSON with information about photos of Dartmouth


```
jsonFlickrApi({
        "photos": {
            "page": 1,
            "pages": 266788,
            "perpage": 10,
            "total": "2667876",
            "photo": [{"id": "5340131446", "secret": "3b7c380bea","server": "5244","farm": 6, ...}
                {"id": "5338762379", "secret": "59f7435b93","server": "5284","farm": 6, ...},
                    ...
                    ...
                ]
        },
        "stat": "ok"
        })
```

Flickr can use JSON to return information about photos it stores

Simplified Flickr JSON 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&format=json

Returns JSON with information about photos of Dartmouth


```
jsonFlickrApi({
        "photos": {
                "page": 1,
                "pages": 266788,
                "perpage": 10,
                "total": "2667876",
                "photo": [{"id": "5340131446", "secret": "3b7c380bea","server": "5244","farm": 6, ...}
                        {"id": "5338762379", "secret": "59f7435b93","server": "5284","farm": 6, ...},
                                ...
                                ...
                        ]
        },
        "stat": "ok"
        })
```

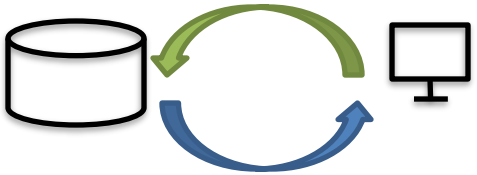

Flickr adds a non-standard header "jsonFlickrApi(" :-|

"photos" object contains "photo" array with information describing each matching photo and where to find it

This information is not the photo itself!
It is how to find the photo on Flickr's servers

Flickr can use JSON to return information about photos it stores

Simplified Flickr JSON 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&format=json

Returns JSON with information about photos of Dartmouth


```
jsonFlickrApi({
        "photos": {
            "page": 1,
            "pages": 266788,
            "perpage": 10,
            "total": "2667876",
            "photo": [{"id": "5340131446", "secret": "3b7c380bea","server": "5244","farm": 6, …}
```


Flickr documentation says that photos can be retrieved with:

https://farm{**farm-id**}.staticflickr.com/{**server-id**}/{**id**}_{**secret**}.jpg

https://farm**6**.staticflickr.com/**5244**/**5340131446**_**3b7c380bea**.jpg

Download actual photo from this web location

Agenda

1. Creating Graphical User Interfaces (GUIs)
2. Getting data from the web
3. Web services
4. Processing data
5. Finished product

FlickrSearchJSON.java: finished product expands upon FlickrSearchCore.java

FlickrSearchJSON.java

Get API key from Canvas (don't abuse it!)

```
24 public class FlickrSearchJSON extends JFrame {
25     private static final int imageWidth = 640, imageHeight = 640;    // medium 640 on flickr
26     private static String api_key = "  Key here                     .";
27
28     private JComponent canvas;                                       // drawing component
29     private JTextField queryF;                                       // GUI text field for query
30     private String sort = "relevance";                               // how to sort when search
31     private ArrayList<BufferedImage> images;                       // loaded images, using Java type
32     private int curr = 0;                                            // index of currently-displayed image
33
34     public FlickrSearchJSON() {
35         super("Flickr Search");
36
37         // Initially no images
38         images = new ArrayList<BufferedImage>();
39
40         // Create our graphics-handling component, sized to hold the images
41         canvas = new JComponent() {
42             public void paintComponent(Graphics g) { //called on repaint()
43                 super.paintComponent(g);
44                 if (images.size() > 0) {
45                     g.drawImage(images.get(curr), 0, 0, null);
46                 }
47             }
48         };
49         canvas.setPreferredSize(new Dimension(imageWidth, imageHeight));
50
51         // Create the GUI components
52         JPanel gui = setupGUI();
53
54         // Put the GUI and the canvas in the panel, one at the top and one taking the rest of the space
55         Container cp = getContentPane();
56         cp.setLayout(new BorderLayout());
57         cp.add(gui, BorderLayout.NORTH);
58         cp.add(canvas, BorderLayout.CENTER);
59
60         // Boilerplate
61         setLocationRelativeTo(null);
62         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
63         pack();
64         setVisible(true);
65     }
```

Will load all Flickr images into ArrayList of BufferedImages called *images*

***curr* will hold the index of image currently displayed**

- **If we have some images (images.size > 0) draw the image at index *curr* in the *canvas* JComponent**
- ***paintComponent()* runs on *repaint()***

FlickrSearchJSON.java: finished product expands upon FlickrSearchCore.java

FlickrSearchJSON.java

```java
67    private JPanel setupGUI() {
68        // prev button steps backward through images
69        JButton prevB = new JButton("prev");
70        prevB.addActionListener(new AbstractAction() {
71            public void actionPerformed(ActionEvent e) {
72                if (images.size() > 0) {
73                    curr--;
74                    if (curr < 0) curr = images.size() - 1;
75                    repaint();
76                }
77            }
78        });
79
80        // prev button steps forward through images
81        JButton nextB = new JButton("next");
82        nextB.addActionListener(new AbstractAction() {
83            public void actionPerformed(ActionEvent e) {
84                if (images.size() > 0) {
85                    curr = (curr + 1) % images.size();
86                    repaint();
87                }
88            }
89        });
```

- **Setup previous graphical button as before, but now add program logic**
- **If "prev" button pressed, go to prior image (loop to last if at image 0)**
- ***repaint()* causes *canvas* to redraw and display the image in ArrayList *images* at index *curr***

- **Next button similar to previous button**

FlickrSearchJSON.java: finished product expands upon FlickrSearchCore.java

FlickrSearchJSON.java

```java
        // sort dropdown (combobox) lists possible ways to sort
        String[] sortOrders = { "relevance", "interestingness-desc", "interestingness-asc",
                "date-taken-desc", "date-taken-asc" };
        JComboBox sortB = new JComboBox(sortOrders);
        sortB.addActionListener(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                sort = (String)((JComboBox)e.getSource()).getSelectedItem();
                System.out.println(sort);
            }
        });

        // text field for the search query
        queryF = new JTextField(20);

        // search button fires off the search
        JButton search = new JButton("search");
        search.addActionListener(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("searching for '" + queryF.getText() + "' by " + sort);
                try {
                    loadImages(queryF.getText());
                    curr = 0;
                    repaint();
                }
                catch (Exception ex) {
                    System.err.println("search failed");
                }
            }
        });

        // Put all the components in a panel
        JPanel gui = new JPanel();
        gui.setLayout(new FlowLayout());
        gui.add(queryF);
        gui.add(sortB);
        gui.add(search);
        gui.add(new JSeparator(SwingConstants.VERTICAL));
        gui.add(prevB);
        gui.add(nextB);
```

- **Setup drop down combo box to track how Flickr should sort photos**
- **Each time drop down changes, *sort* instance variable updates**

- **When "search" button clicked, get search text in *queryF* JTextField**
- **Then call *loadImages* method passing query text from *queryF* to get images from Flickr (next slide)**
- **Set current image to 0 and *repaint()***

FlickrSearchJSON.java: finished product expands upon FlickrSearchCore.java

FlickrSearchJSON.java

Search query entered by user

```
143    private void loadImages(String query) throws Exception {
144        // Get rid of existing images
145        images.clear();
146
147        // Build the REST query as specified in the Flickr API
148        String request = "https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=" + api_key +
149                "&text=" + URLEncoder.encode(query,"UTF-8") + "&sort=" + sort + "&per_page=10&format=json";
150        System.out.println("search:" + request);
151
152        // Read JSON response from Flickr and store in String str
153        BufferedReader in = new BufferedReader(new InputStreamReader(new URL(request).openStream()));
154        String str = "", line;
155        while ((line = in.readLine()) != null) str += line;
156
157        //strip out Flickr's annoying extra text "jsonFlickrApi(" and closing ")" so that we have valid json
158        str = str.substring("jsonFlickrApi(".length(), str.length()-1);
159        System.out.println(str);
160
161
162        try {
163            //parse flickr's response as JSON
164            JSONParser parser = new JSONParser();
165            JSONObject jsonFlickrResponse = (JSONObject) parser.parse(str); //parse in the string returned by Flickr as json
166
167            //get photo array from photos object
168            JSONObject photosJsonObject = (JSONObject) jsonFlickrResponse.get("photos"); //get photos json object from Flickr response
169            JSONArray photosList = (JSONArray) photosJsonObject.get("photo"); //now we have a List with information about photos
170
171            //loop over each photo in photo array
172            for (int i=0;i<photosList.size(); i++) {
173                //get each photo
174                JSONObject photoDetails = (JSONObject) photosList.get(i);
175                try {
176                    // Build the image URL as specified in the Flickr API
177                    String url = "http://farm" + photoDetails.get("farm") + ".staticflickr.com/" +
178                            photoDetails.get("server") + "/" + photoDetails.get("id") + "_" +
179                            photoDetails.get("secret") + "_z.jpg"; //_z means size=640
180                    System.out.println(photoDetails.get("title") + " - " + url);
181
182                    //fetch image at URL and add to images
183                    BufferedImage img = ImageIO.read(new URL(url));
184                    images.add(img);
185
186                }
187                catch (IOException e) {
188                    System.out.println("couldn't load image");
189                }
190            }
191        }
```

Build *request* URL with query string parameters

Use URLEncoder to handle spaces in String *query*

Create new BufferedReader and read Flickr's response to *request,* just like reading a file

Get rid of Flickr's annoying header, to get standard JSON

Parse JSON and get list of photos

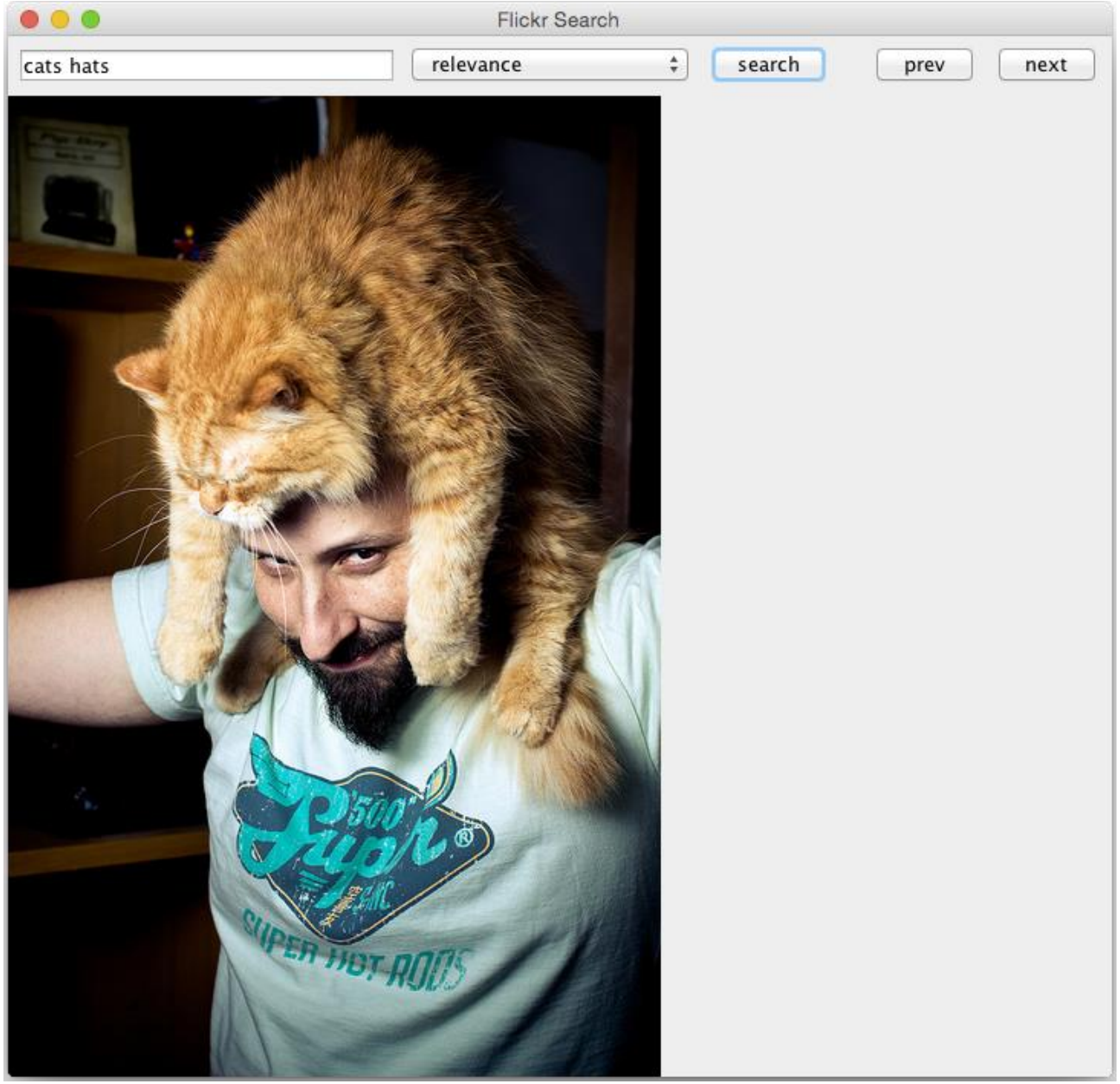
JSONArrays are Lists

JSONObjects are Maps

Loop over all photos returned using List photosList

Extract farm, server, and secret data elements about each photo from Map

Fetch photo and add to images ArrayList


Flickr Search
cats hats
relevance
search
prev
next
SUPER HOT RODS

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

Sample XML for course enrollment

Start of enrollment tag begins with "<"

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

End of enrollment tag begins with "</"

XML

- **XML groups data with an opening and closing tag**

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

Sample XML for course enrollment

Start of enrollment tag begins with "<"

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

Nested tag called "course" for CS 1

End of enrollment tag begins with "</"

XML

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

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

Sample XML for course enrollment

Start of enrollment tag begins with "<"

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

Nested tag called "course" for CS 1

Another nested tag called "course" for CS 10

End of enrollment tag begins with "</"

XML

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

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

Sample XML for course enrollment

Course tag attributes: department = “CS”, number = 1, term = “18W”

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

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

Sample XML for course enrollment

Course tag attributes: department = “CS”, number = 1, term = “18W”

```
<enrollment>
    <course department="CS" number="1" term=”18W">
        <student name="Alice" year="20" />
        <student name="Bob" year="19" />
        <student name="Charlie" year="18" />
    </course>
    <course department="CS" number="10" term=”18W">
        <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 of representing data

Sample XML for course enrollment

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

FlickrSearchXML.java: finished product expands upon FlickrSearchCore.java

FlickrSearch.java

```
165    private void loadImages(String query) throws Exception {
166        // Get rid of existing images
167        images.clear();
168
169        // Build the REST query as specified in the Flickr API
170        String request = "https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=" + api_key +
171                "&text=" + URLEncoder.encode(query,"UTF-8") + "&sort=" + sort + "&per_page=10";
172        System.out.println("search:" + request);
173
174        // Get the XML document as a string
175        //BufferedReader in = new BufferedReader(new FileReader("inputs/test.xml"));
176        BufferedReader in = new BufferedReader(new InputStreamReader(new URL(request).openStream()));
177        String xml = collectString(in);
178
179        // Parse XML, following Oracle example
180        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
181        DocumentBuilder builder = factory.newDocumentBuilder();
182        InputSource source = new InputSource();
183        source.setCharacterStream(new StringReader(xml));
184        Document doc = builder.parse(source);
185
186        // Loop over all photo elements
187        NodeList photos = doc.getElementsByTagName("photo");
188        for (int i = 0; i < photos.getLength(); i++) {
189            Node n = photos.item(i);
190            try {
191                // Build the image URL as specified in the Flickr API
192                String url = "http://farm" + attribute(n, "farm") + ".staticflickr.com/" +
193                        attribute(n, "server") + "/" + attribute(n, "id") + "_" +
194                        attribute(n, "secret") + "_z.jpg"; //_z means size=640
195                System.out.println(attribute(n, "title") + " - " + url);
196
197                //fetch image at URL and add to images
198                BufferedImage img = ImageIO.read(new URL(url));
199                images.add(img);
200            }
201            catch (IOException e) {
202                System.out.println("couldn't load image");
203            }
204        }
205    }
206
```

Start with search query entered by user

Build *request* URL with query string parameters

Use URLEncoder to handle spaces in String *query*

- Create new BufferedReader and read Flickr's response to *request*
- Clean up non-standard XML in *collectString()* – this is a hack!

Follow Oracle's example to set up XML parser

Loop over all photos returned

Extract farm, server, and secret data elements about each photo

Fetch photo and add to images ArrayList