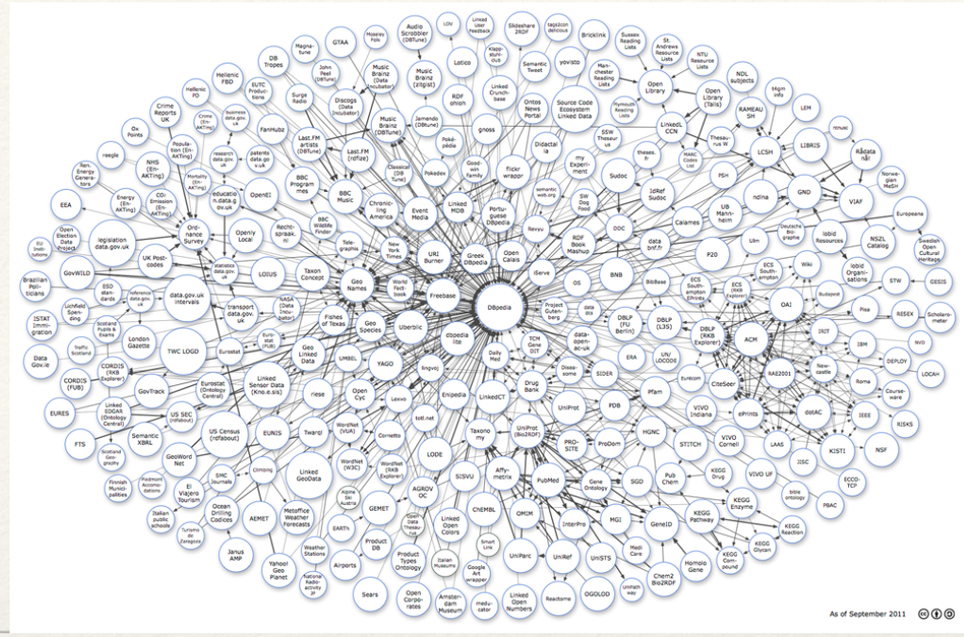
Dr. Sarah Abraham
University of Texas at Austin
Computer Science Department



Data Structures and Formats

Elements of Graphics
CS324e

Arrays

- Structure that can hold any object of a declared type
  - `int[] values = new int[10];`
- Index accesses data stored within the array
- Length of array has a fixed size upon array creation
  - `values.length;`

ArrayLists

- Structure of mutable length that can hold any object of a declared type
  - `ArrayList<Type> values = new ArrayList<Type>();`
- Size of ArrayList adapts as data is added or removed
  - `values.add(object);`
  - `values.remove(index);`
- `get(index)` returns the object stored at index
- `set(index, object)` sets the value of the object stored at index
- `size()` returns the length of the ArrayList

Lists

- ArrayList provides flexibility for object creation at the cost of Array efficiency
- FloatList, IntList and StringList are specialized ArrayLists that are both flexible and efficient
  - Only work with floats, ints and Strings respectively
- `append(value)` adds value (of given type) to the List
- Other convenience features can be found within Processing website's Reference

Dictionaries

- Provide key-value pairs for efficient lookup
  - Key is a String
  - Value is the data
- FloatDict, IntDict and StringDict provide dictionary lookup for floats, ints and Strings respectively
- `set("key", value)` adds the key-value pair to the dictionary
- `get("key")` retrieves the value assigned to "key"
- `hasKey("key")` returns a boolean whether or not "key" is part of the dictionary

HashMaps

- Provide a more flexible key-value pair than Dicts
- Key and value can be of any type
- Access functionality with `import java.util.Map;`
- Constructor assigns key-value pair upon object creation
  - `HashMap<String, Integer> hm = new HashMap<String, Integer>();`
- `put(key, value)` adds key-value pair to HashMap
- `get(key)` retrieves the value associated with key

HashMap Functionality

- Not native to Processing — built from Java library
- Reference at: http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html

Objects

- Also a type of data structure!
- Some objects are defined in Processing (String, PShape, PImage, etc)
- Other objects are defined by the program-specific classes or libraries of classes
- When should you create an Object class versus store object data in one of the pre-existing containers?

Data Storage and Formatting

- Input data must be parsed (separated) and loaded into a program before it is usable
- Documents provide a consistent style of storage between programs
  - Aids parsing
  - Aids human-readability
- Processing supports manipulation of three document types:
  - XML
  - JSON
  - CSV (Table)

CSV

- Storage for tabular data
- Provides plain text representation of information
- Row is data record comprised fields
- Fields store necessary information
- Flat storage structure

Note: Whitespace in Data

- Invisible characters that provide visual formatting
- Tabs delimit text within a line (\t)
- Newlines delimit text by line (\n)
- Assist in data separation in plain text files

CSV Examples

- Surnames
- Starbucks locations

Tables (CSVs)

- Processing's Table class loads data as rows and columns
- `loadTable(filename, options);`
  - `"header"` option automatically interprets header row
  - Options `"tsv"` and `"csv"` allow interpretation of .txt files with tab-separated or comma-separated values

Accessing Tables

- `getRowCount()` returns the number of rows
- `getColumnCount()` returns the number of columns
- `getInt()`, `getFloat()` and `getString()` take two parameters:
  - `getInt(row, column); //Takes row and column index`
  - `getInt(row, columnName); //Takes row index and column name (based on header)`

Accessing Rows

- `rows()` accesses all rows of a Table
- TableRows can also call `getInt()`, `getFloat()` and `getString()` methods

```
for (TableRow r : t.rows()) {
 String id = r.getString(“object”);
 float x = r.getFloat(“x”);
 float y = r.getFloat(“y”);
 }
```

More Table Methods

- `findRows(value, column)` finds all rows that contain the value in given column (id or name)
- `matchRows(regexp, column)` finds all rows that match the regular expression in given column (id or name)
- `addRow()` creates a new TableRow in the Table
- `removeRow(index)` removes the TableRow at index in the Table

Instapoll Question: Tables

- What are the table formats Processing accepts?

Hands-on: Using CSVs

- Today's activities:
  1. Find a csv file from a site like this: https://vincentarelbundock.github.io/Rdatasets/datasets.html
  2. Load in the table data and access each row individually
  3. Store these values in an internal data structure (e.g. your own custom class)
  4. Represent these values in some way on the screen

XML

- Markup language based on open standards
- Designed for information transport
- Markup provides document structure
- Content contains desired information
- Tags define logical components called elements
  - Elements can nest
  - Tags are not pre-defined — user-decided

XML

- Tags defined by angle brackets
- Content placed within tags
- Tags can be nested

```
<element>
 <item>First item</item>
 <item>Second item</item>
 </element>
```

- Nested elements are children of the enclosing elements

XML Examples

- RSS feed (http://feeds.bbci.co.uk/news/rss.xml?edition=uk)
- eBay auction data (http://www.cs.washington.edu/research/xmldatasets/data/auctions/ebay.xml)

XML in Processing

- XML is Processing's built-in XML parser
- `loadXML(filename)` loads `filename` as element of an XML tree
  - Can also load from web address that references an XML file (e.g. RSS feeds)
- Parsed XML object will contain all children objects within that element

Accessing XML Elements

- `getChildren("element")` returns an array of XML objects that are the `element` children
  - `XML[] children = root.getChildren("element");`
- Possible to iterate over all rows

```
for (XML c : root.getChildren("element")){}
```

Parsing Elements

- `getChild("element")` returns the child element labeled `element`
- `getContent()` returns all content within the element
  - `getFloatContent()` and `getIntContent()` return float and int content
- `getInt()`, `getFloat()` and `getString()` return attributes within an element

JSON

- Notation for attribute-value pairs of transmitted data
- Designed to replace XML
- Objects are collections of attribute-value pairs
- Arrays provide an ordered list of objects
- Objects can represent hierarchies

JSON

- Model for objects and arrays
- Like XML presents hierarchical structures
- Often easier to structure and parse

- Objects are unordered name/value pairs: `{name1: value1, name2: value2}`
- Arrays are order collections of values: `[value1, value 2]`
- Objects and arrays can be values

JSON Examples

- Zip codes
- Company information

JSON in Processing

- JSON arrays stored within JSONArray
  - `loadJSONArray(filename)` loads JSON `filename`
- JSON objects stored within JSONObjects
  - `loadJSONObject(filename)` loads JSON `filename`
- `getJSONObject(index)` returns the JSON object at index within JSONArray

Accessing JSON Objects

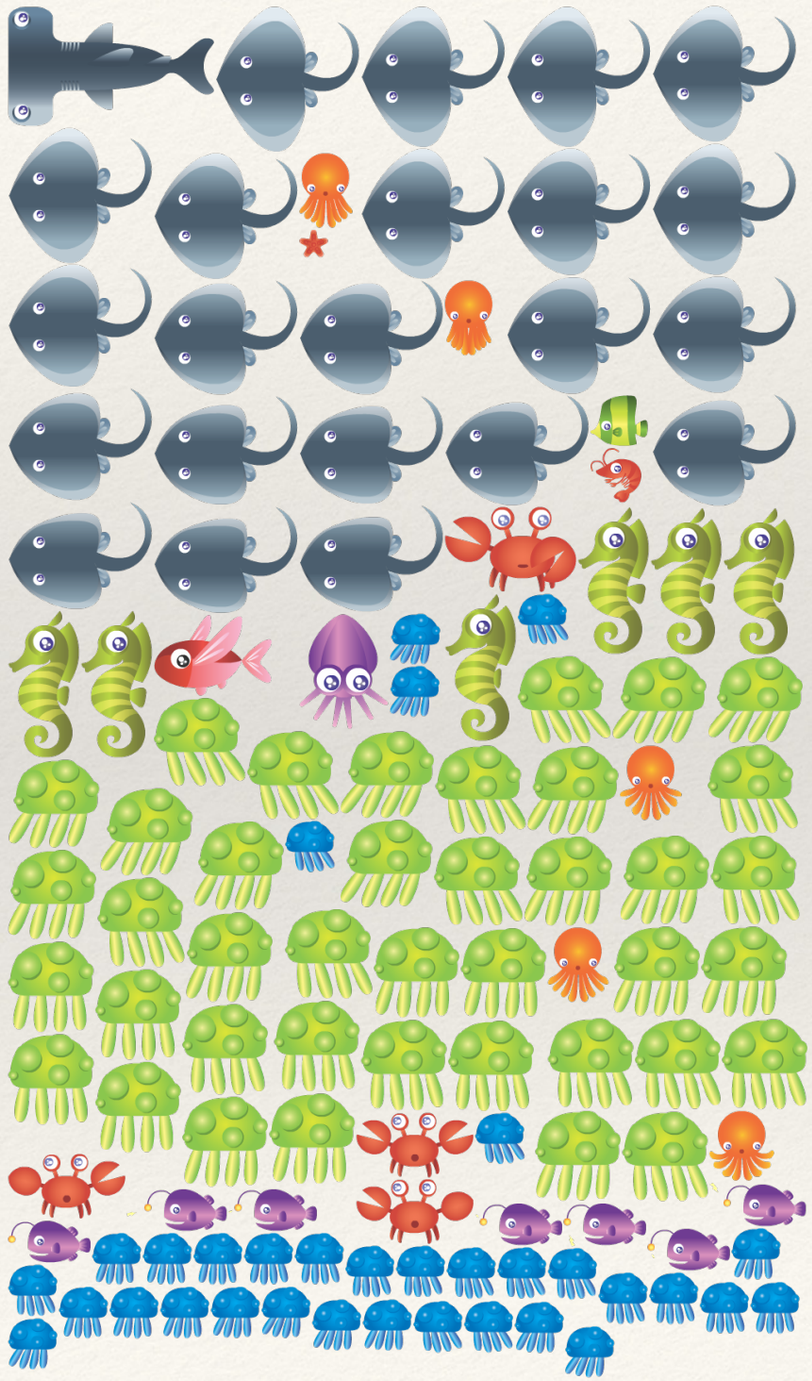
- `getInt("attribute")`, `getFloat("attribute")` and `getString("attribute")` return the value associated with an object's attribute
- `setInt(index, value)`, `setFloat(index, value)` and `setString(index, value)` overwrite values at JSONArray index (or JSONObject attribute)
- `append(value)` appends any value (int, float, String, JSONObject, JSONArray etc) to the JSONArray
- `remove(index)` removes value at index in JSONArray

Sprite Sheet Redux

- Sprite sheets require an image with densely packed sprites along with **metadata** for pulling out the correct sprite
- Metadata often stored as XML or JSON
  - Name of sprite
  - Position of sprite
  - Orientation of sprite
  - Size of sprite
  - Trim around sprite
  - etc...

Sprite Sheet Example

- { "filename": "starfish0000", "frame": {"x":370,"y":284,"w":36,"h":36}, "rotated": false, "trimmed": false, "spriteSourceSize": {"x":0,"y":0,"w":36,"h":36}, "sourceSize": {"w":36,"h":36} } ,
- { "filename": "stingray0000", "frame": {"x":2,"y":329,"w":182,"h":154}, "rotated": false, "trimmed": true, "spriteSourceSize": {"x":0,"y":16,"w":182,"h":184}, "sourceSize": {"w":182,"h":184} }



Hands-on: Using Data Formats

- Today's activities:

1. Access the RSS data from this site (http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/data/auctions/ebay.xml), and import it into Processing
2. Experiment with `getChild()` and `getContent()` methods to extract the data into custom classes
3. Access JSON data from this site (https://adobe.github.io/Spry/samples/data_region/JSONDataSetSample.html#Example1), and import it into Processing
4. Experiment with `getJSONObject()` and other JSON `get()` methods to extract the data into custom classes