

Parsing JSON, Using Libraries, Java Collections, Generics

Grading (subject to change)

- **Code Reviews (50%)**
- **Final OpenFrameworks (C++) Project (30%) or (40%)**
- **Participation (20%)**
 - 10% iClickers (8% attendance, 2% performance)
 - 10% short assignments online and in class

JSON (www.json.org)

- **JavaScript Object Notation**
- **A lightweight data-interchange format**
 - Very commonly used by APIs
- **It is easy for humans to read and write.**
- **It is easy for machines to parse and generate.**

Example JSON object

```
{  
  name_of_a_string: "a string",  
  name_of_a_number: 2080.8827,  
  objects_can_be_values: { here_is: "another object" },  
  an_array: [ 27, "word", { objects_can: "be in arrays" } ]  
}
```

Using APIs (e.g., <https://newsapi.org>)

- API = Application Programming Interface
- Get an API key
- Grab some JSON:
 - https://newsapi.org/v1/articles?source=associated-press&sortBy=top&apiKey=YOUR_API_KEY_HERE
- JSON formatter/pretty printer
 - <https://jsonformatter.curiousconcept.com>
 - There are a bunch of these, use your favorite

Parsing JSON in Java

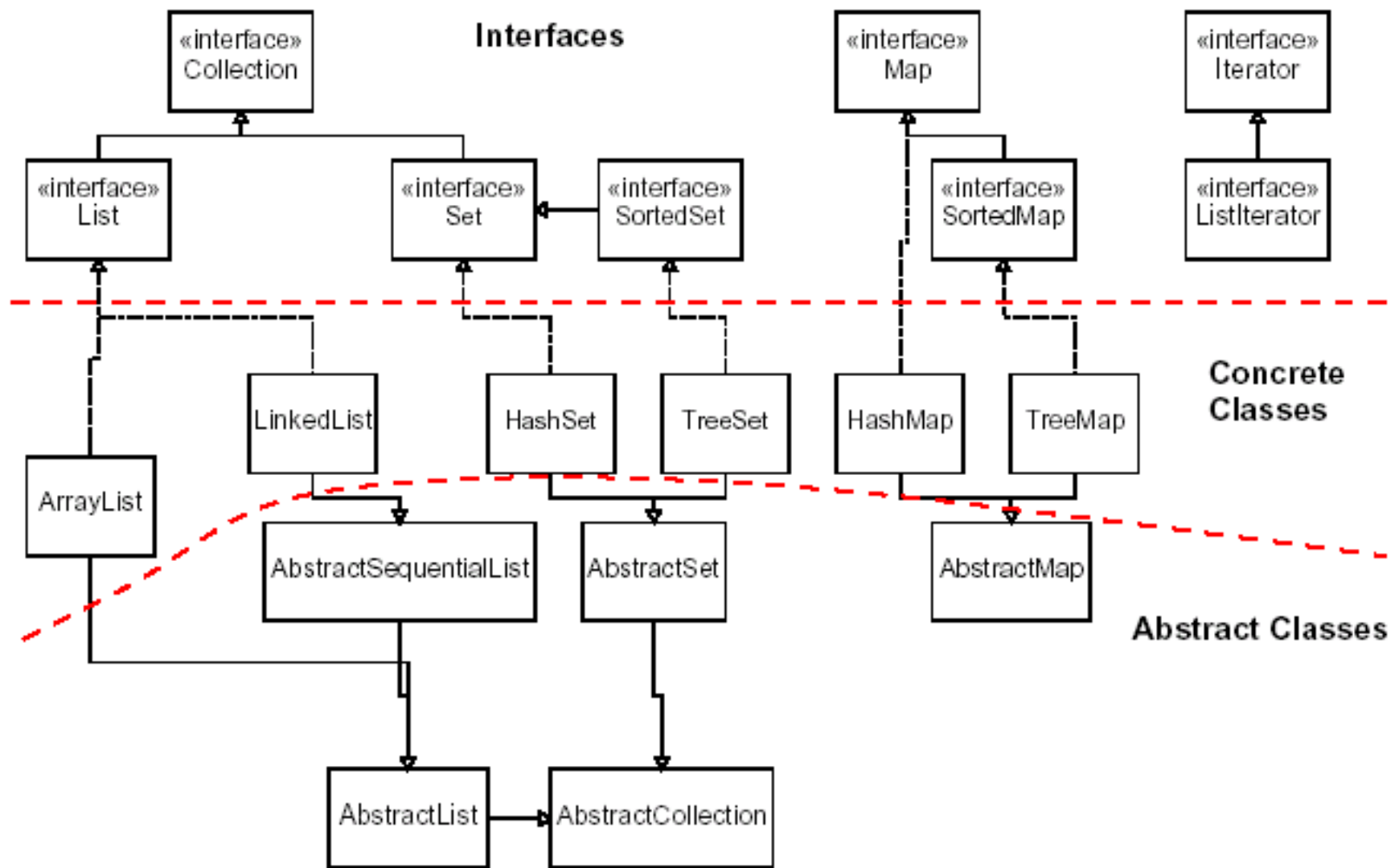
- **Use the GSON library from Google**
 - <https://github.com/google/gson/blob/master/UserGuide.md>
 - Use Maven to add the library to your project
- **Build classes with fields for the desired elements of the JSON**
 - Use the same names and get the types right
- **Instantiate a Gson object**
 - `Gson gson = new Gson();`
- **Use the fromJSON method to parse the JSON**
 - `Thing newThing = gson.fromJson(jsonString, Thing.class);`
 - `Thing [] thingArray = gson.fromJson(jsonString, Thing[].class);`
- **Extended example using NewsAPI**

Java Collections

- **collection: an object that stores data; a.k.a. "data structure"**
 - the objects stored are called **elements**
 - some collections maintain an ordering; some allow duplicates
 - typical operations: *add, remove, clear, contains* (search), *size*
- examples found in the Java class libraries:
 - `ArrayList, HashMap, TreeSet`
- all collections are in the `java.util` package

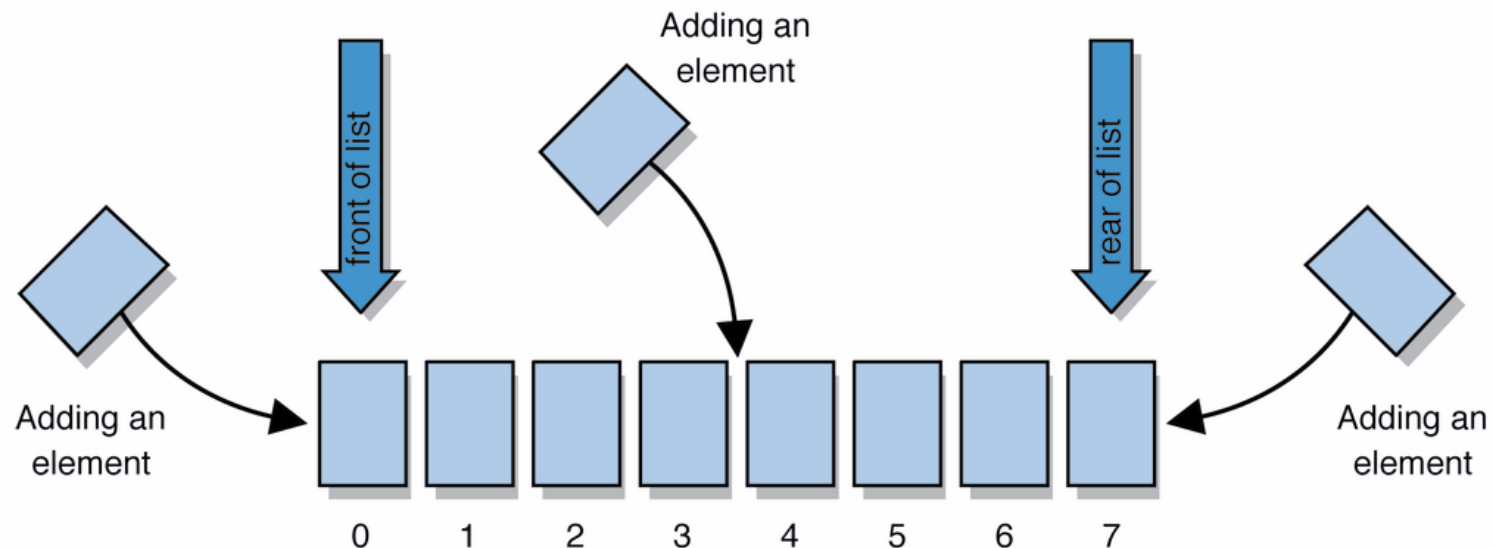
```
import java.util.*;
```

Java Collection Framework



Lists

- **list: a collection storing an ordered sequence of elements**
 - each element is accessible by a 0-based **index**
 - a list has a **size** (number of elements that have been added)
 - elements can be added to the front, back, or elsewhere
 - in Java, a list can be represented as an **ArrayList** object



ArrayList Methods (partial list)

<code>add (value)</code>	appends value at end of list
<code>add (index, value)</code>	inserts given value just before the given index, shifting subsequent values to the right
<code>clear ()</code>	removes all elements of the list
<code>indexOf (value)</code>	returns first index where given value is found in list (-1 if not found)
<code>get (index)</code>	returns the value at given index
<code>remove (index)</code>	removes/returns value at given index, shifting subsequent values to the left
<code>set (index, value)</code>	replaces value at given index with given value
<code>size ()</code>	returns the number of elements in list
<code>toString ()</code>	returns a string representation of the list such as "[3, 42, -7, 15]"

Map

- Allows lookups from one kind of object to find another object

```
Map<KeyType, ValueType> myMap =  
    HashMap<KeyType, ValueType>();
```

```
KeyType key = ...;
```

```
ValueType value = ...;
```

```
myMap.put(key, value);
```

```
ValueType lookup = myMap.get(key);
```

```
assert lookup == value;
```

Map Interface

- `put(k, v)` Associate **v** with **k**
- `get(k)` The value associated with **k**

- `size()` The number of pairs
- `isEmpty()` Whether it is empty

- `remove(k)` Remove the mapping for **k**
- `clear()` Remove all mappings

- `containsKey(k)` Whether contains a mapping for **k**
- `containsValue(v)` Whether contains a mapping to **v**

Generics

```
ArrayList<Type> name = new ArrayList<Type>();
```

- When constructing an `ArrayList`, you must specify the type of elements it will contain between `<` and `>`.
 - This is called a *type parameter* or a *generic* class.
 - Allows the same `ArrayList` class to store lists of different types.
 - Must be objects (vs. primitive types)

Boxed Primitive Types

- Can't do `ArrayList<int>`
- Java provides “boxed primitives”: E.g., `Integer`
 - Sub-class of object
- Can do:
 - `ArrayList<Integer> lengths = new ArrayList<Integer>`
 - `lengths.add(7);` // automatically promoted to boxed type

Primitive Type	Wrapper Type
<code>int</code>	<code>Integer</code>
<code>double</code>	<code>Double</code>
<code>char</code>	<code>Character</code>
<code>boolean</code>	<code>Boolean</code>