Parsing JSON, Using Libraries, Java Collections, Generics

Slides adapted from Craig Zilles

CamelCaser Difficulty

How difficult was the first assignment?

A. **Easy**
B. **Moderate**
C. **Challenging**
D. **Unreasonable**

CamelCaser Time

How long did it take you to complete the assignment?

A. **Less than 2 hours**
B. **2 to 4 hours**
C. **4 to 6 hours**
D. **6 to 8 hours**
E. **More than 8 hours**

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**

What if we want to filter News Articles?

- E.g., only select those articles with specific authors
- What should be the return type of such a function?

One Implementation

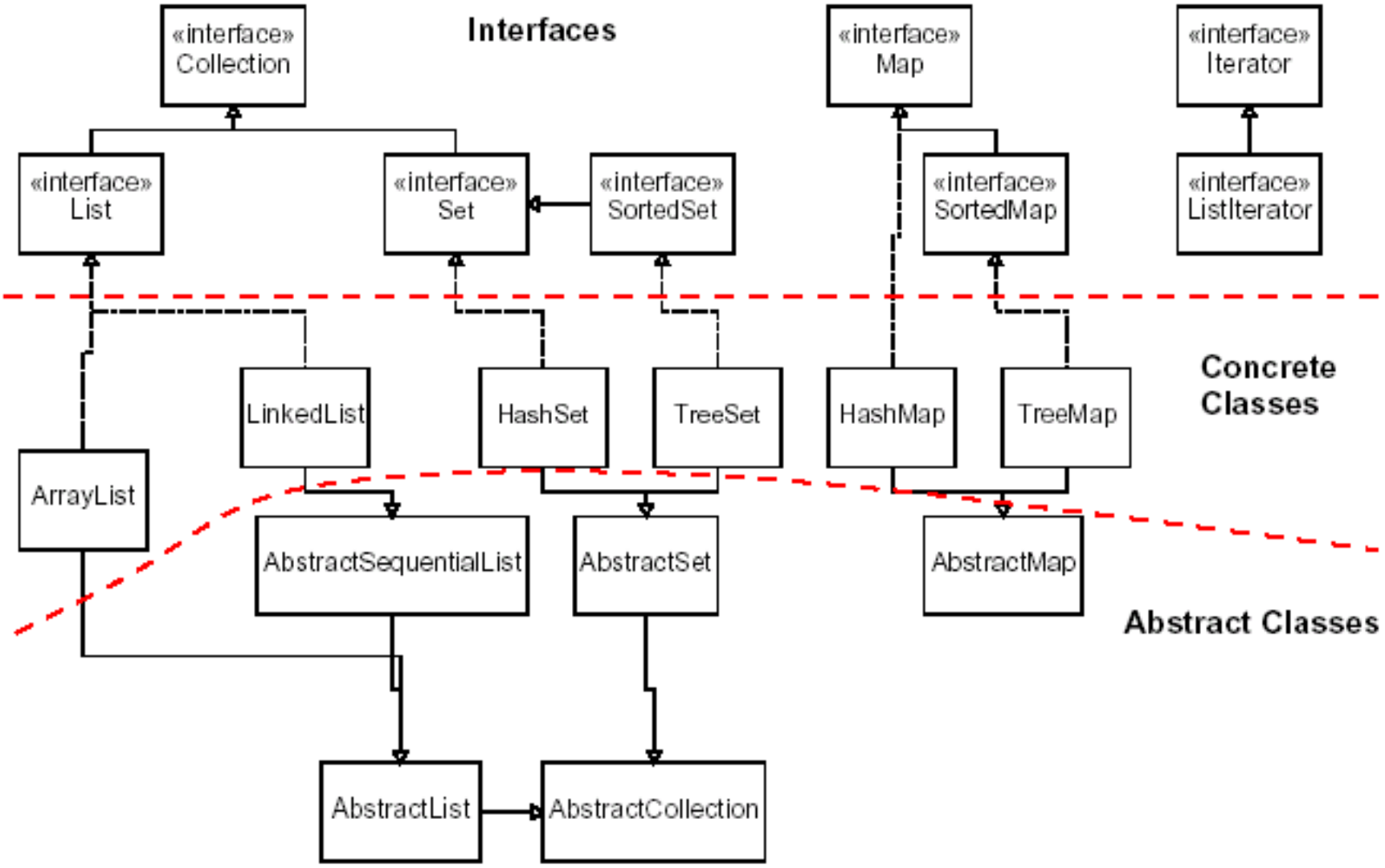
```
public NewsArticle[]
removeNullAuthorArticles(NewsArticle[] input) {
    // output array can't be bigger than input array
    NewsArticle [] output = new NewsArticle[input.length];
    int outputIndex = 0;

    for (int i = 0; i < input.length; i++) {
        if (input[i].getAuthor() != null) {
            output[outputIndex] = input[i];
            outputIndex ++;
        }
    }

    return output;
}
```

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

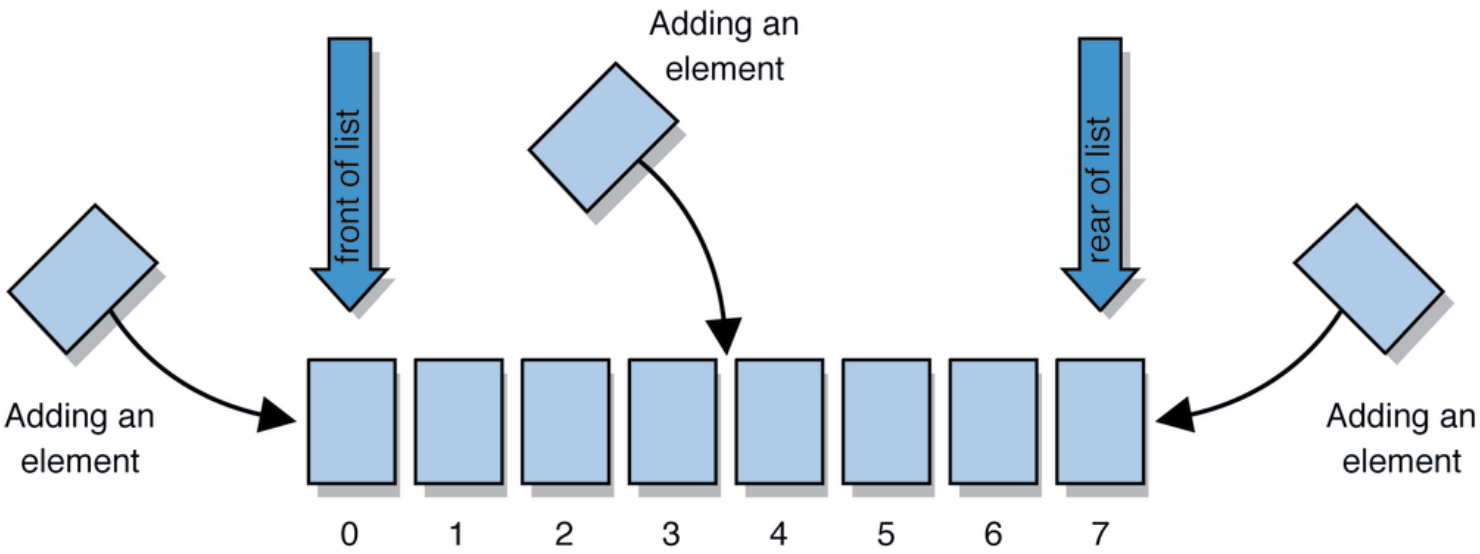
```
import java.util.*;
```

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)

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

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
`int`	`Integer`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

To Dos for Next Tuesday

- **Read chapter 4 of your book “Aesthetics”**
- **Read section 4 (Formatting) of the Google Java Style Guide**
- **Take Policy Quiz**
- **Assignment for next week’s code review:**
  - Parsing JSON for UofI course grade distributions
  - Filtering and aggregating data from this sources
  - Out now