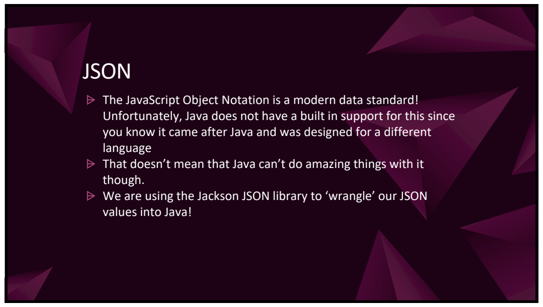
Java JSON IO

CS 1410 - 2024
Cody Henrichsen

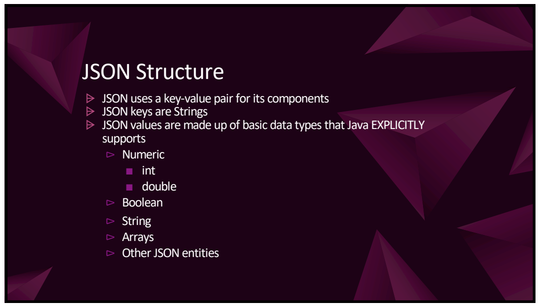
1



JSON

- The JavaScript Object Notation is a modern data standard! Unfortunately, Java does not have a built in support for this since you know it came after Java and was designed for a different language
- That doesn't mean that Java can't do amazing things with it though.
- We are using the Jackson JSON library to 'wrangle' our JSON values into Java!

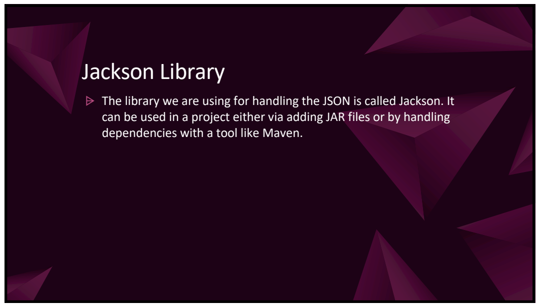
2



JSON Structure

- JSON uses a key-value pair for its components
- JSON keys are Strings
- JSON values are made up of basic data types that Java EXPLICITLY supports
  - Numeric
    - int
    - double
  - Boolean
  - String
  - Arrays
  - Other JSON entities

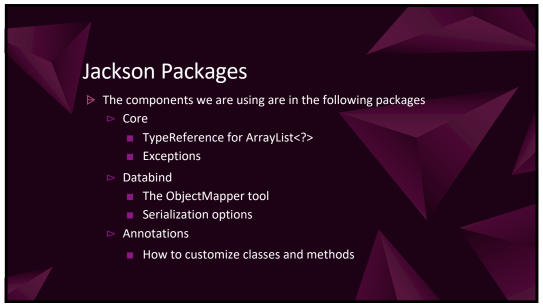
3



Jackson Library

- The library we are using for handling the JSON is called Jackson. It can be used in a project either via adding JAR files or by handling dependencies with a tool like Maven.

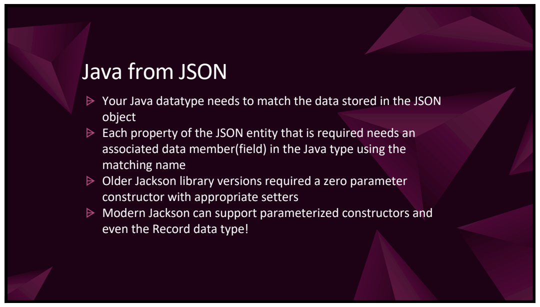
4



Jackson Packages

- The components we are using are in the following packages
  - Core
    - TypeReference for ArrayList<?>
    - Exceptions
  - Databind
    - The ObjectMapper tool
    - Serialization options
  - Annotations
    - How to customize classes and methods

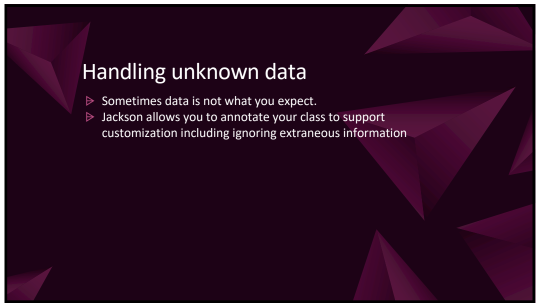
5



Java from JSON

- Your Java datatype needs to match the data stored in the JSON object
- Each property of the JSON entity that is required needs an associated data member(field) in the Java type using the matching name
- Older Jackson library versions required a zero parameter constructor with appropriate setters
- Modern Jackson can support parameterized constructors and even the Record data type!

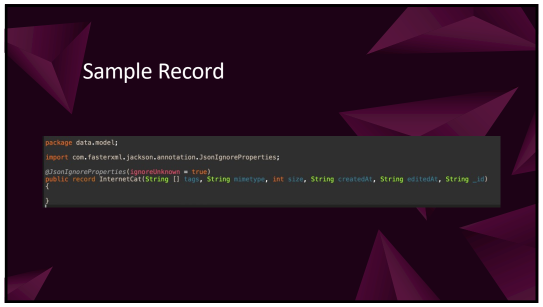
6



Handling unknown data

- Sometimes data is not what you expect.
- Jackson allows you to annotate your class to support customization including ignoring extraneous information

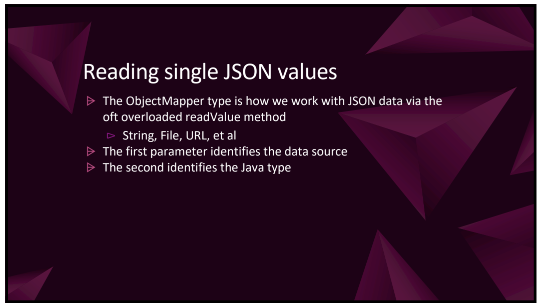
7



Sample Record

```java
package data.model;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@JsonIgnoreProperties(ignoreUnknown = true)
public record InternetCat(String [] tags, String mimetype, int size, String createdAt, String editedAt, String _id)
{

}
```

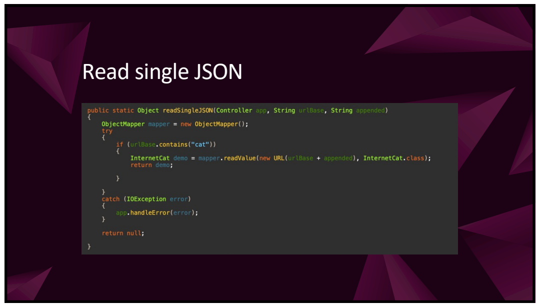
8



Reading single JSON values

- The ObjectMapper type is how we work with JSON data via the oft overloaded readValue method
  - String, File, URL, et al
- The first parameter identifies the data source
- The second identifies the Java type

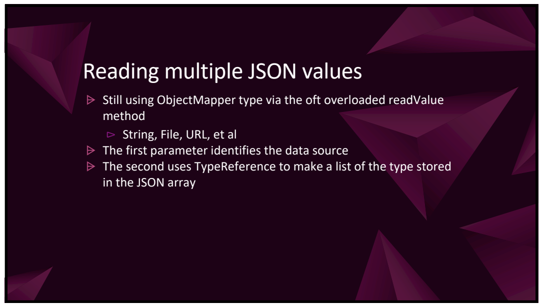
9



Read single JSON

```
public static Object readSingleJSON(Controller app, String urlBase, String appended)
{
    ObjectMapper mapper = new ObjectMapper();
    try
    {
        if (urlBase.contains("cat"))
        {
            InternetCat demo = mapper.readValue(new URL(urlBase + appended), InternetCat.class);
            return demo;

        }

    }
    catch (IOException error)
    {
        app.handleError(error);
    }

    return null;

}
```

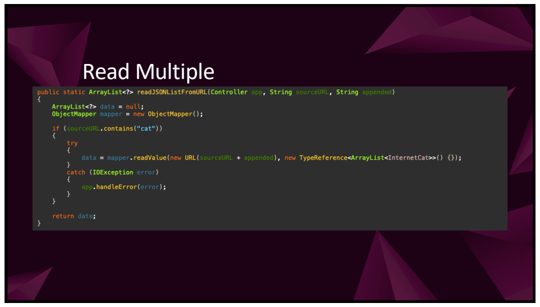
10



Reading multiple JSON values

- Still using ObjectMapper type via the oft overloaded readValue method
  - String, File, URL, et al
- The first parameter identifies the data source
- The second uses TypeReference to make a list of the type stored in the JSON array

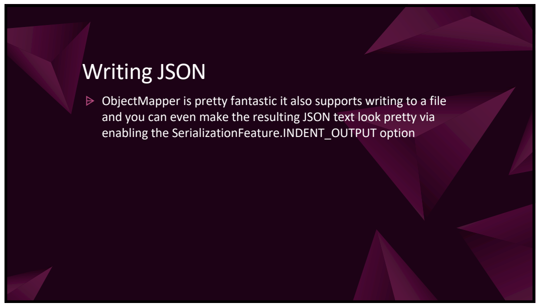
11



Read Multiple

```
public static ArrayList<?> readJSONListFromURL(Controller app, String sourceURL, String appended)
{
    ArrayList<?> data = null;
    ObjectMapper mapper = new ObjectMapper();

    if (sourceURL.contains("cat"))
    {
        try
        {
            data = mapper.readValue(new URL(sourceURL + appended), new TypeReference<ArrayList<InternetCat>>() {});
        }
        catch (IOException error)
        {
            app.handleError(error);
        }
    }

    return data;
}
```

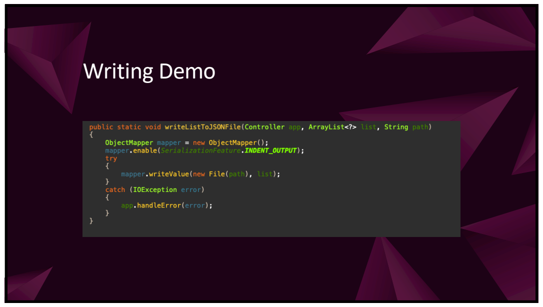
12



Writing JSON

- ObjectMapper is pretty fantastic it also supports writing to a file and you can even make the resulting JSON text look pretty via enabling the SerializationFeature.INDENT_OUTPUT option

13



Writing Demo

```
public static void writeListToJSONFile(Controller app, ArrayList<?> list, String path)
{
    ObjectMapper mapper = new ObjectMapper();
    mapper.enable(SerializationFeature.INDENT_OUTPUT);
    try
    {
        mapper.writeValue(new File(path), list);
    }
    catch (IOException error)
    {
        app.handleError(error);
    }
}
```

14