

Additional Background for Project 1

JavaScript Object Notation (JSON)

- JavaScript Object Notation (JSON) is a standard format for storing and transmitting structured data.
- Core data structures include:
 - Objects, to store unordered collections of key-value pairs
 - {"key1": value1, "key2": value2, ...}
 - Arrays, to store an ordered list of values
 - [value1, value2, ...]
 - Object and array values can be of any type, and it is common to have nested objects.
- View JSON objects using <https://jsonviewer.stack.hu/>

Sample JSON

```
[  
  {  
    "code": "202710",  
    "description": "Fall 2026 (View Only)"  
  },  
  {  
    "code": "202650",  
    "description": "Summer 2026 (View Only)"  
  }  
]
```

Reading JSON data into Java

- In order to read the JSON data into Java, we can apply **data binding** to automatically map a JSON string directly to a Java object based on matching property names.
- This can be done with a Java **record**

```
/**  
 * Term record to store term information  
 * @param code the term code  
 * @param description the term description  
 */  
@JsonIgnoreProperties(ignoreUnknown = true)  
public record Term(String code, String description) {}
```

Java Record

- A Java record is a special type of class for storing immutable data.
- Java automatically generates
 - private, final fields for record components (e.g., code and description)
 - a public constructor that initializes all fields
 - Getter-like methods that match the field names exactly (e.g., *Term.code()*)
 - Additional methods such as *equals*

Reading JSON into Java

- We use an *ObjectWrapper* from the Jackson library to read in JSON data
- *TypeReference* is required is required when using generics

```
/**
 * Parses the JSON content of available terms into an ArrayList of terms
 * @param content The JSON string representing the available terms.
 * @return an ArrayList of maps, where each map represents a term
 *         with its code and description
 * @throws IOException if terms cannot be parsed
 */
private ArrayList<Term> parseTerms(String content) throws IOException {
    ObjectMapper mapper = new ObjectMapper();
    ArrayList<Term> terms = mapper.readValue(
        content, new TypeReference<ArrayList<Term>>() {}
    );
    return terms;
}
```

Working with files

- Use *Files.readString* and *Files.writeString* to read from and write to a file

```
// create a Path object for accessing the file
Path cacheFile = Path.of("cache/terms.txt");
```

```
// read content of file into String if file exists
String content;
if (Files.exists(cacheFile)) {
    content = Files.readString(cacheFile);
    return content;
}
```

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/nio/file/Files.html>