Colton Kohler

5/18/2025

M11: Programming Assignment

**Jackson: A Java JSON API Overview**

Jackson is a widely-used open-source library for processing JSON in Java. Often referred to as “the best JSON parser for Java,” Jackson goes beyond simple parsing to provide a comprehensive suite of tools for working with data on the Java platform. It supports not only JSON but also various other data formats such as XML, CSV, BSON, YAML, Avro, Smile (binary JSON), and more. This makes it extremely flexible and valuable for developers building applications with complex data-handling requirements.

Jackson is composed of several core modules: jackson-core, jackson-annotations, and jackson-databind. The jackson-core module provides the streaming API for parsing and generating JSON. jackson-annotations allows developers to add metadata to Java classes to control how they are serialized or deserialized. The jackson-databind module handles automatic data binding between Java objects and JSON, meaning developers can easily convert Java objects to JSON and vice versa using the ObjectMapper class.

One of Jackson’s key strengths is its modular architecture. Developers can extend it with additional modules for specific data types or formats. For example, the Jackson team provides modules for supporting Java 8 features (jackson-datatype-jsr310, jackson-module-parameter-names), Joda-Time, Guava collections, and more. Jackson also supports schema generation and validation, offering tools for generating JSON Schema documents directly from Java class definitions.

Jackson’s development is active and robust. As of 2025, the project has three major version lines: 1.x (deprecated), 2.x (currently stable and widely used), and the in-progress 3.x. Each version line uses different Java packages and artifact IDs, which allows for compatibility across versions. According to the project maintainers, “a project can depend on both Jackson 1.x and 2.x (and eventually 3.x), without conflicts” (Jackson GitHub, 2025).

Developers are encouraged to use Maven Central to download Jackson. All Jackson modules are published there. The main JAR for the core streaming functionality (jackson-core) is available at:

https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core

The ObjectMapper class is the most commonly used entry point. Here’s a basic example of serializing and deserializing with Jackson:

ObjectMapper mapper = new ObjectMapper();

String json = mapper.writeValueAsString(myObject);

MyObject obj = mapper.readValue(json, MyObject.class);

Jackson’s flexibility, performance, and extensive documentation make it a popular choice among Java developers.

**Additional Java JSON Example: Gson**

An alternative to Jackson is the Gson library developed by Google. Gson allows Java developers to convert Java objects into JSON and back. It is particularly appreciated for its simplicity and its ability to work with classes without requiring annotations or modifications to source code.

Gson supports generic types, custom serializers, and complex object graphs. Unlike many other libraries, Gson was designed to allow serialization and deserialization of objects for which you do not have source code access. According to the maintainers, “Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of” (Gson GitHub, 2025).

Here is a simple example of using Gson:

Gson gson = new Gson();

String json = gson.toJson(myObject);

MyObject obj = gson.fromJson(json, MyObject.class);

Gson can be downloaded via Maven Central here:

https://mvnrepository.com/artifact/com.google.code.gson/gson

While Gson is currently in maintenance mode and not actively adding major new features, it remains a reliable choice for many use cases.

**Conclusion**

Both Jackson and Gson offer powerful solutions for JSON handling in Java. Jackson’s modularity and extended data format support make it a strong choice for more complex or large-scale applications. Gson’s simplicity and ease of use make it ideal for smaller projects or when working with external or legacy code. Each library has its strengths, and developers can choose based on the requirements of their specific projects.

**References**

FasterXML. (2025). Jackson GitHub Repository. https://github.com/FasterXML/jackson

Google. (2025). Gson GitHub Repository. https://github.com/google/gson

Maven Repository. (n.d.). Jackson-core. https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core

Maven Repository. (n.d.). Gson. https://mvnrepository.com/artifact/com.google.code.gson/gson