M Ahmad 21L5780
Hassan Tariq I215784
Umair Azeem I216091
Ahmad Khakan I216108

CODE OVERVIEW

GSON MAIN

Top-Level Classes (Core Public API)

These are the most important user-facing classes/interfaces in Gson.

File	Purpose
Gson.java	**Main entry point** for serialization/deserialization. Has toJson() and fromJson() methods.
GsonBuilder.java	Used to **configure and build** a Gson instance (e.g., register custom adapters, policies).
JsonElement.java	Abstract superclass for JsonObject, JsonArray, JsonPrimitive, and JsonNull. Represents a JSON value.
JsonObject.java, JsonArray.java, JsonPrimitive.java, JsonNull.java	Implementations of JsonElement. Map to JSON object, array, string/number/boolean, and null.
JsonParser.java	Parses raw JSON strings into a JsonElement. Deprecated in favor of Gson.fromJson().

JsonStreamParser.java	Parses multiple JSON elements from a stream of JSON data.
JsonIOException, JsonSyntaxException, JsonParseException	Exception types for JSON parsing and writing issues.
JsonSerializer, JsonDeserializer	Interfaces for **custom serialization/deserialization** logic.
JsonSerializationContext, JsonDeserializationContext	Passed to custom (de)serializers to help (de)serialize fields.
ExclusionStrategy, FieldAttributes	Control which fields should be excluded from (de)serialization.
FieldNamingStrategy, FieldNamingPolicy	Defines how Java field names map to JSON keys (e.g., camelCase to snake_case).
InstanceCreator	Allows defining how to instantiate a class when default constructor is not available.
LongSerializationPolicy	Defines how longs should be serialized (as number or string).
ToNumberPolicy, ToNumberStrategy	Control how numeric values are parsed (e.g., as Double, Long, etc.).
FormattingStyle, Strictness, ReflectionAccessFilter	Control fine-grained settings like output style, access filters, parsing behavior.

---

annotations/ Package

Contains custom annotations supported by Gson.

Annotation	Purpose

@SerializedName	Rename a field in JSON.
@Expose	Allows selective field exposure.
@JsonAdapter	Register a specific adapter for a field/type.
@Since, @Until	Include field only if version is within range.

---

internal/ Package

Contains **internal helpers** not meant for public use. Crucial for understanding the machinery.

File	Purpose
ConstructorConstructor.java	Builds instances of objects via reflection or unsafe hacks.
ObjectConstructor.java	Interface for creating objects.
Excluder.java	Applies @Expose, @Since, @Until, ExclusionStrategy rules.
UnsafeAllocator.java	Allows Gson to construct objects **without no-arg constructor** using low-level hacks.
Streams.java	Reads/writes JsonElement using JsonReader/JsonWriter.
GsonTypes.java, Primitives.java	Handle Java type operations, including generics and wrappers.
LinkedTreeMap.java	Gson's own Map implementation that preserves order like LinkedHashMap.

LazilyParsedNumber.java	Used for deferring number parsing until needed.
NumberLimits.java	Validates number sizes.
GsonPreconditions.java	Internal assertions/checks.
ReflectionAccessFilterHelper.java, JsonReaderInternalAccess.java	Control internal reflective access.

---

internal/bind/ Package

Houses **TypeAdapters and Factories** – the heart of serialization/deserialization.

File	Purpose
TypeAdapters.java	Contains **default adapters** for primitives, strings, etc.
ReflectiveTypeAdapterFactory.java	Builds adapters by **reflecting over fields** – core of Gson's default behavior.
ArrayTypeAdapter.java, CollectionTypeAdapterFactory.java, MapTypeAdapterFactory.java	Handle arrays, lists, and maps respectively.
JsonElementTypeAdapter.java	Serializes/deserializes JsonElement instances.
ObjectTypeAdapter.java	Fallback adapter that tries to guess type dynamically.
TreeTypeAdapter.java, TypeAdapterRuntimeTypeWrapper.java	Allow dynamic delegation and runtime type resolution.
JsonAdapterAnnotationTypeAdapterFactory.java	Factory for @JsonAdapter support.

SerializationDelegatingTypeAdapter.java	Delegates to another adapter during serialization only.
DefaultDateTypeAdapter.java	Handles java.util.Date and custom date formats.
EnumTypeAdapter.java	Handles enum values.

---

reflect/ Package

File	Purpose
TypeToken.java	Captures full generic type info (e.g., List<Foo>). Crucial for handling generics.
ReflectionHelper.java	Assists with field/method access via reflection.

---

stream/ Package

Low-level JSON streaming parser and writer (used internally and exposed via JsonReader/Writer).

File	Purpose
JsonReader.java	**Pull-based** JSON reader – similar to a tokenizer.
JsonWriter.java	Low-level JSON writer. Used by Gson.toJson().
JsonToken.java	Enum for current JSON token type (e.g., BEGIN_OBJECT).
JsonScope.java	Internal class to track reader state.
MalformedJsonException.java	Thrown if JSON is invalid during reading.

---

internal/sql/ Package

File	Purpose
SqlDateTypeAdapter.java, SqlTimestampTypeAdapter.java, SqlTimeTypeAdapter.java	Type adapters for SQL date/time types.
SqlTypesSupport.java	Checks for SQL type support availability.

GSON EXTRA

extras/examples/rawcollections/

RawCollectionsExample.java

- Demonstrates how **raw types** (like Map or List without generics) can be deserialized.
- Shows type safety issues and how Gson deals with them.
- Useful for understanding Gson's behavior when generic type info is erased.

---

graph/

GraphAdapterBuilder.java

- Supports **serialization of object graphs with cycles** (circular references).
- Builds TypeAdapterFactory to handle identity-based serialization/deserialization.
- Useful for cases like: Person → Address → Person
- Tests: GraphAdapterBuilderTest.java confirms cycle handling works correctly.

---

interceptors/

Gson doesn't support interceptors by default — this is an **extension mechanism** for pre-/post-processing JSON.

File	Purpose
Intercept.java	Annotation to mark methods to run after deserialization.
InterceptorFactory.java	Scans for @Intercept and injects logic into the deserialization process.
JsonPostDeserializer.java	Applies post-deserialization hooks. Acts like a middleware layer.

Example use case: Validating or enriching data *after* deserialization (e.g., fixing nulls, running a validator, setting defaults).

---

typeadapters/

These are **custom TypeAdapterFactories** for advanced behaviors.

RuntimeTypeAdapterFactory.java

- Allows **polymorphic deserialization** (e.g., handling subclasses based on a type field).

Use case:

json
CopyEdit
{ "type": "credit_card", "number": "1234" }

-
    - Deserializes to CreditCard if type == "credit_card".

PostConstructAdapterFactory.java

- Looks for methods annotated with @PostConstruct (like in Java EE).
- Automatically invokes them after the object is deserialized.
- Can be used for object initialization or validation.

UtcDateTypeAdapter.java

- A strict date adapter for **UTC ISO 8601** date formats.
- Ensures consistent date parsing/formatting across time zones.

---

test/

Contains JUnit tests validating the behavior of:

- GraphAdapterBuilder

- Interceptor logic
- Runtime type and post-construction adapters
- UTC date serialization

Each test file mirrors the structure and name of the corresponding implementation class.

Summary of Use Cases

Feature	Use Case
GraphAdapterBuilder	Cyclic graphs / object identity
RuntimeTypeAdapterFactory	Polymorphic types based on a discriminator field
PostConstructAdapterFactory	Run init logic after deserialization
InterceptorFactory + @Intercept	Data validation or transformation after deserialization
UtcDateTypeAdapter	Reliable date parsing with UTC

GSON METRICSS

You're now inside the **Gson metrics module**, which is a performance benchmarking suite. It's used to evaluate **serialization and deserialization speed** of Gson (and sometimes compare it with other libraries like Jackson).

Purpose of gson/metrics

Goal: Measure performance of Gson using various JSON structures and usage patterns (e.g., flat vs. nested objects, collections, streaming).

This module uses **benchmark test cases** to track:

- Time taken to serialize objects
- Time taken to deserialize objects
- Streaming vs. tree vs. binding vs. skipping vs. other modes
- Performance on real-world data (e.g., large JSON feeds in ParseBenchmarkData.zip)

---

src/main/java/com/google/gson/metrics/

BagOfPrimitives.java

- A **simple POJO** (Plain Old Java Object) used as a test subject.
- Contains fields like int, long, boolean, String.
- Used in multiple benchmarks as a lightweight test case.

BagOfPrimitivesDeserializationBenchmark.java

- Benchmark for deserializing BagOfPrimitives.
- Helps measure overhead of field reflection, parsing, and object creation.

CollectionsDeserializationBenchmark.java

- Benchmarks deserializing **collections** like List<BagOfPrimitives>.
- Tests Gson's ability to handle generic types and multiple objects efficiently.

SerializationBenchmark.java

- Measures **serialization performance** of different data structures.
- Compares output size and time taken.

ParseBenchmark.java

This is the **most comprehensive benchmark class**, and it includes:

- Nested static classes representing a **real-world feed format** (like RSS or Twitter feeds):
    - Feed, Item, User, Tweet, Link, etc.
- Multiple parsing strategies:
    - GsonDomParser: uses JsonElement (DOM)
    - GsonStreamParser: uses JsonReader
    - GsonSkipParser: skips unnecessary parts
    - GsonBindParser: uses POJO binding
    - JacksonBindParser, JacksonStreamParser: for comparing with Jackson

Insight: This is where Gson and Jackson are compared head-to-head.

NonUploadingCaliperRunner.java

- A modified runner for **Google Caliper**, a microbenchmarking framework.
- Runs the benchmark **locally only** (no uploading to Caliper servers).

---

resources/ParseBenchmarkData.zip

- Contains **real JSON datasets** for benchmarks.
- Used in ParseBenchmark to simulate parsing feeds.

Output: target/

- Compiled .class files for all benchmarks
- gson-metrics-2.13.2-SNAPSHOT.jar: packaged benchmark code (not the main Gson lib)

Summary Table

File	Purpose
BagOfPrimitives	Simple test POJO
SerializationBenchmark	Gson performance during serialization

CollectionsDeserializationBenchmark	Gson performance deserializing collections
ParseBenchmark	Heavy benchmark with feed data, various parsing modes
NonUploadingCaliperRunner	Runs Caliper benchmarks locally
Jackson*Parser	Allows comparison with Jackson
ParseBenchmarkData.zip	Real-world JSON data for testing

PROTO

Purpose of gson/proto

> Enable JSON ↔ Protobuf support via Gson.

This is **not** part of the core Gson library — it's an **extension** that provides:

- A TypeAdapter for Protobuf messages.
- Support for Protobuf annotations like @JsonName.
- Handling of Protobuf's primitive types, enums, nested messages, and repeated fields.

Key Files and Their Roles

ProtoTypeAdapter.java

- This is the **core adapter** that:

    - Serializes Protobuf messages to JSON using field names or annotations.
    - Deserializes JSON back into Protobuf messages using reflection.
- Supports:
    - Custom field name strategies
    - Enum serialization (name or ordinal)
    - Nested messages and lists (repeated fields)

---

src/test/java/com/google/gson/protobuf/functional/

These are **functional tests** to verify integration between Protobuf and Gson:

Test File	Purpose
ProtosWithPrimitiveTypesTest.java	Tests basic scalar fields like int, float, bool, etc.
ProtosWithComplexAndRepeatedFieldsTest.java	Tests **nested messages** and repeated (array) fields.
ProtosWithAnnotationsTest.java	Tests Protobuf messages with @GsonFieldNamingPolicy, @SerializedName, etc.
ProtosWithAnnotationsAndJsonNamesTest.java	Tests Protobuf's json_name property in .proto files.

---

src/test/proto/

annotations.proto

- Defines Protobuf message types with custom annotations (e.g., json_name, custom options).
- Helps validate how well Gson honors Protobuf metadata.

bag.proto

- Defines a variety of Protobuf message types:
    - Simple message
    - Nested message
    - Messages with repeated fields
    - Messages with different case formats

These are compiled into .java classes (see generated-sources/protobuf) using the protoc compiler.

---

generated-test-sources/protobuf/java/...

These are auto-generated Java classes based on .proto files:

- Classes like Bag.ProtoWithAnnotations, Bag.SimpleProto, etc.
- Follow the typical structure:
    - Message, Builder, OrBuilder interfaces
    - Support Protobuf's serialization API

Used in unit tests to test Gson's ability to parse/serialize Protobuf messages to/from JSON.

---

protoc-dependencies/ and protoc-plugins/

Folder	Purpose
protoc-dependencies	Contains standard Protobuf .proto files (like timestamp.proto, struct.proto) required to compile more complex messages.
protoc-plugins	Contains the protoc executable used to generate Java files from .proto definitions.

---

surefire-reports/

Contains output from test executions:

- .txt and .xml files show test results for Protobuf integration (passed/failed/stack traces).

Summary

Area	Description
ProtoTypeAdapter.java	Core Gson adapter for Protobuf messages
.proto files	Define sample Protobuf message structures
functional/ tests	Validate serialization/deserialization of real Protobuf messages
generated/ sources	Auto-generated Java classes from .proto for testing

protoc-dependencies	Core .proto files used in Protobuf ecosystem
protoc-plugins	Local Protobuf compiler (protoc)

Refactorings Report

High-Level Architecture

GSON's software structure is designed for modularity and extension, leveraging object-oriented features and runtime reflection in Java. Below are the key building blocks:

1. Gson Class (Facade)

- Main entry point for JSON operations.
- Provides APIs like toJson() and fromJson().
- Encapsulates complexity of type handling.

2. Type Adapters (TypeAdapter<T>)

- Contains logic to serialize and deserialize specific Java types.
- Implements a variation of the Strategy pattern.

3. Type Adapter Factories (TypeAdapterFactory)

- Generates TypeAdapter instances based on types.
- Supports extension and plug-in capabilities.

4. Reflection & Type Resolution

- Dynamically processes type information at runtime.
- Uses TypeToken<T> to handle generics.

5. JSON Stream API

- Utilizes JsonReader and JsonWriter for low-level parsing.
- Ensures memory efficiency through stream-based processing.

6. Internal Helper Modules

- Includes ConstructorConstructor, Streams, etc.
- Facilitates instantiation and type resolution.

Source Code Organization

- com.google.gson: Core engine classes.
- com.google.gson.reflect: Tools for generic type inspection.
- com.google.gson.stream: Streaming interface for JSON.
- com.google.gson.internal: Utility logic and reflection support.
- com.google.gson.internal.bind: Built-in adapter implementations.

Design Guidelines Followed

1. **Open/Closed Principle** – Extendable via new adapters without modifying existing logic.
2. **Separation of Concerns** – Parsing, type resolution, and I/O separated.
3. **DRY (Don't Repeat Yourself)** – Utility modules avoid code duplication.
4. **Fail-Fast Behavior** – Detects and reports parsing issues early.

Applied Design Patterns

Design Pattern	Application
Facade	Unified API via the Gson class

Strategy	Custom logic through TypeAdapter<T>
Factory	Adapter instantiation
Decorator	Enhancing or wrapping adapters
Reflection	Dynamic field and type handling

Implementation Quality Concerns

Common Issues Identified Across Modules

- **Verbose Anonymous Classes**: Increase boilerplate and reduce clarity.
- **Broad Exception Handling**: Suppresses specific error context.
- **Silent Fallback Behavior**: Errors ignored during parsing and reflection.
- **Unsafe Raw Types**: Use of unchecked generics prone to runtime errors.
- **Lack of Input Validation**: Especially for custom labels and reflection methods.
- **Inconsistent Naming**: Reduces clarity and readability.
- **Obsolete Tools**: Use of deprecated runners and legacy benchmarking frameworks.
- **Hardcoded Data**: Reduces reusability in tests and benchmarks.

Protos Module

- Long method (deserialize()) with deep logic nesting.
- Repetition in enum handling.
- Over-coupling to FieldDescriptor.
- Direct embedding of reflection.
- Complex conditionals.
- Manual control flow around enum strategies.

GSON-Metrics Module

- Use of public fields.
- Lack of toString() for core classes.
- Deprecated benchmarking tools.
- Hardcoded sample data.
- Missing cleanup in benchmark teardown.

GSON-Extras Module

- Multiple redundant DateFormat instances.
- Thread safety issues with shared formatters.
- Poor error handling: catch-all blocks.
- Weak validation in serialization factories.
- Unsafe casting and unchecked types.

Improvement Actions Taken

In Protos

- Reorganized ProtoTypeAdapter into methods and utilities.
- Introduced helper classes: EnumValueResolver, FieldNameResolver, Utils.
- Extracted reflection logic.
- Refined deserialization process and enum handling.

In GSON-Metrics

- Introduced encapsulation via private fields and accessors.
- Removed hardcoded data.
- Added toString() methods.
- Added teardown support to benchmark tests.

In GSON-Extras

- Switched to ThreadLocal<SimpleDateFormat>.
- Introduced structured error handling and logging.
- Replaced ad-hoc JSON creation with GSON writers.
- Introduced input validation and stronger type safety.

Outcomes from Code Refinement

1. **Improved Separation of Concerns**
    - Components are modular and focused.
2. **Enhanced Maintainability**
    - Clear logic separation and utility reuse.
3. **Stronger Type Safety**
    - Elimination of raw and unchecked generics.
4. **Reliable Error Handling**
    - Logging and contextual exception capture.
5. **Improved Readability**
    - Clear naming, documentation, and responsibility decomposition.

Summary

The analysis of GSON revealed structural inefficiencies, design inconsistencies, and code complexity issues across modules. The remediation work involved decomposition of large components, consolidation of utility logic, application of defensive coding techniques, and a shift toward type-safe practices. These efforts enhanced the clarity, stability, and extensibility of the GSON codebase, ensuring its continued reliability and evolution.

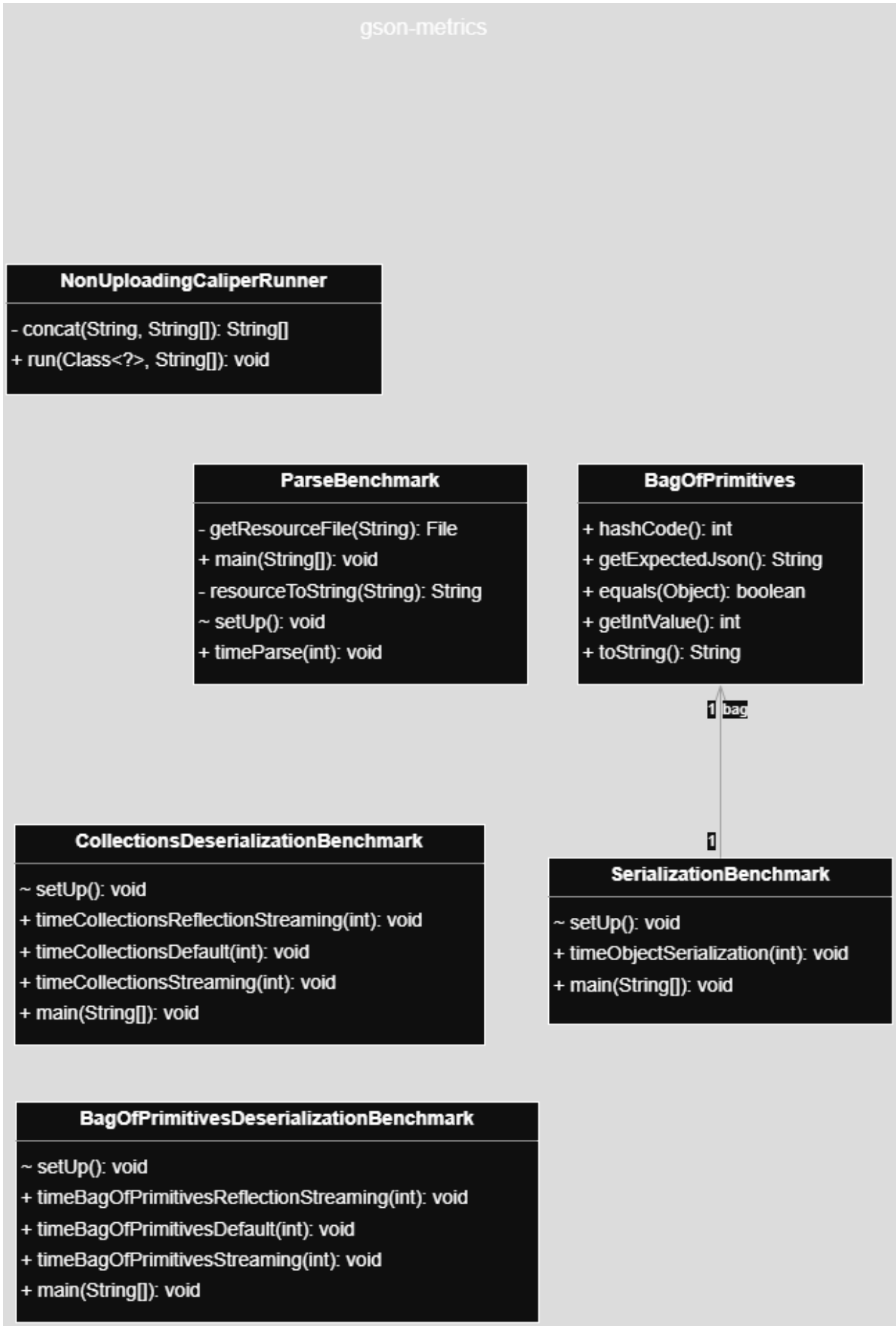
GSON MAIN
https://app.diagrams.net/?&highlight=0000ff&edit=_blank&layers=1&nav=1&title=exported_from_idea.drawio#G1k-1zn_b836x1TM3P8cKGNMxx8WGFa62s#%7B%22pageId%22%3A%2204mfruOeYYT_trqJg_yq%22%7D

GSON EXTRA
https://app.diagrams.net/?&highlight=0000ff&edit=_blank&layers=1&nav=1&title=exported_from_idea.drawio#G1k-1zn_b836x1TM3P8cKGNMxx8WGFa62s#%7B%22pageId%22%3A%2204mfruOeYYT_trqJg_yq%22%7D


gson-extra
PostConstructAdapterFactoryTest
+ testList(): void
+ test(): void
<<annotation>>
Intercept
+ postDeserialize(): Class<JsonPostDeserializer>
InterceptorFactory
+ create(Gson, TypeToken<T>): TypeAdapter<T>
RawCollectionsExample
+ main(String[]): void
UtcDateTypeAdapter
+ write(JsonWriter, Date): void
- parse(String, ParsePosition): Date
- format(Date, boolean, TimeZone): String
- padInt(StringBuilder, int, int): void
- parseInt(String, int, int): int
+ read(JsonReader): Date
- checkOffset(String, int, char): boolean
GraphAdapterBuilder
+ registerOn(GsonBuilder): void
+ addType(Type, InstanceCreator<?>): GraphAdapterBuilder
+ addType(Type): GraphAdapterBuilder
UtcDateTypeAdapterTest
+ testWellFormedParseException(): void
+ testLocalTimeZone(): void
+ testDifferentTimeZones(): void
+ testUtcWithJdk7Default(): void
+ testUtcDatesOnJdkBefore1_7(): void
+ testNullDateSerialization(): void
<<interface>>
JsonPostDeserializer
+ postDeserialize(T): void
InterceptorTest
+ setUp(): void
+ testExceptionsPropagated(): void
+ testCollection(): void
+ testDirectInvocationOfTypeAdapter(): void
+ testList(): void
+ testTopLevelClass(): void
+ testMapKeyAndValues(): void
+ testCustomTypeAdapter(): void
+ testField(): void
RuntimeTypeAdapterFactory
+ registerSubtype(Class<T>, String): RuntimeTypeAdapterFactory<T>
+ of(Class<T>, String, boolean): RuntimeTypeAdapterFactory<T>
+ of(Class<T>): RuntimeTypeAdapterFactory<T>
+ create(Gson, TypeToken<R>): TypeAdapter<R>
+ of(Class<T>, String): RuntimeTypeAdapterFactory<T>
+ recognizeSubtypes(): RuntimeTypeAdapterFactory<T>
+ registerSubtype(Class<T>): RuntimeTypeAdapterFactory<T>
GraphAdapterBuilderTest
+ testSerializeListOfLists(): void
+ testAddTypeCustomInstanceCreator(): void
+ testDeserialization(): void
+ testAddTypeOverwrite(): void
+ testDeserializeListOfLists(): void
+ testDeserializationWithMultipleTypes(): void
+ testDeserializationDirectSelfReference(): void
+ testSerializationWithMultipleTypes(): void
+ testBuilderReuse(): void
+ testSerialization(): void
RuntimeTypeAdapterFactoryTest
+ testRuntimeTypeIsBaseType(): void
+ testRuntimeTypeAdapter(): void
+ testDuplicateLabel(): void
+ testDeserializeMissingTypeField(): void
+ testNullSubtype(): void
+ testSerializeCollidingTypeFieldName(): void
+ testNullTypeFieldName(): void
+ testDuplicateSubtype(): void
+ testNullBaseType(): void
+ testDeserializeMissingSubtype(): void
+ testRuntimeTypeAdapterRecognizeSubtypes(): void
+ testSerializeWrappedNullValue(): void
+ testNullLabel(): void
+ testSerializeMissingSubtype(): void
PostConstructAdapterFactory
+ create(Gson, TypeToken<T>): TypeAdapter<T>

GSON METRICS

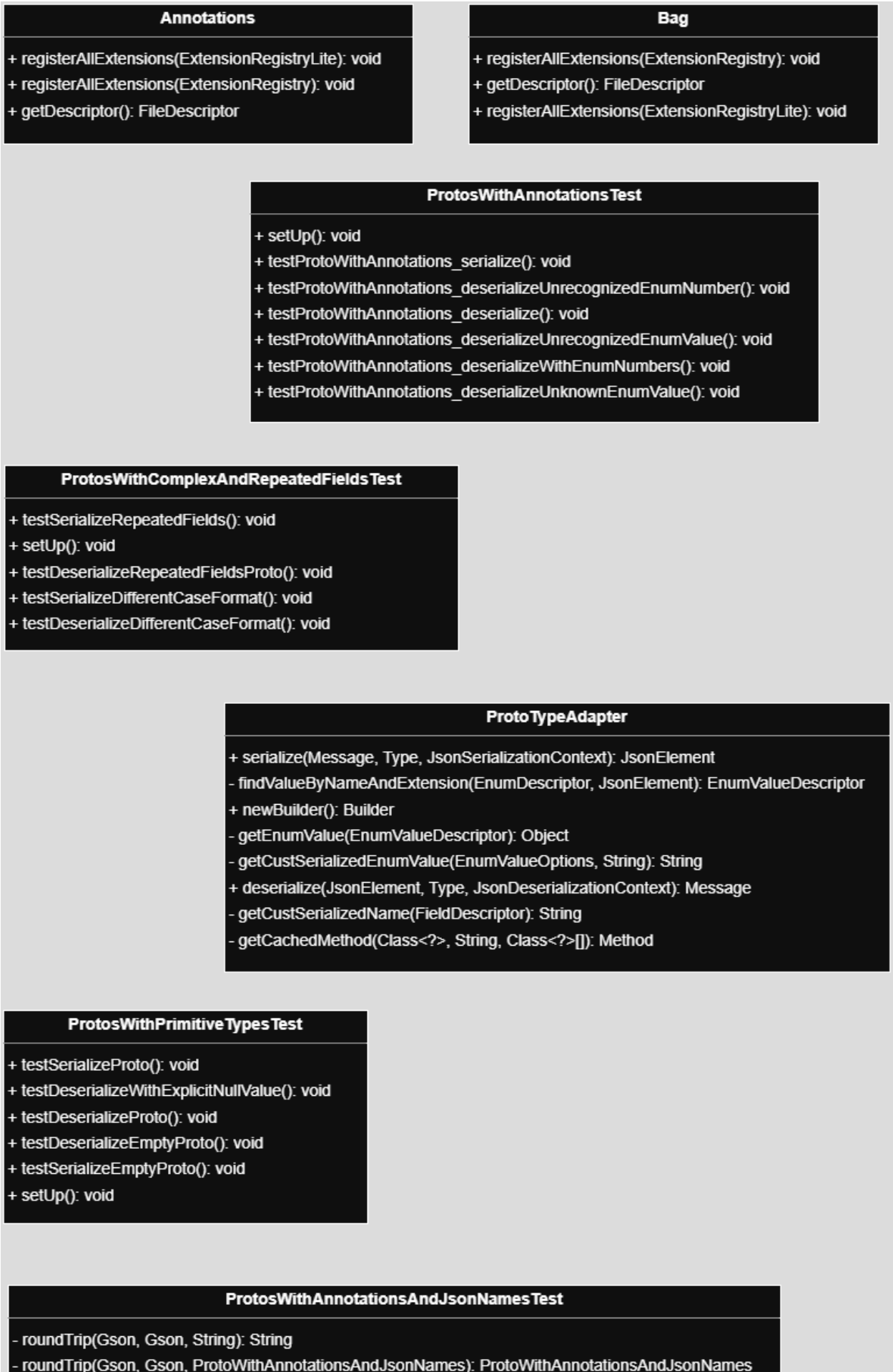
https://app.diagrams.net/?&highlight=0000ff&edit=_blank&layers=1&nav=1&title=exported_from_idea.drawio#G1yq81o6FP6SgGzitE40jTKQumlYlvkcgy#%7B%22pageId%22%3A%22v2GGWXU58Qwtb11Mqw6o%22%7D


gson-metrics
NonUploadingCaliperRunner
- concat(String, String[]): String[]
+ run(Class<?>, String[]): void
ParseBenchmark
- getResourceFile(String): File
+ main(String[]): void
- resourceToString(String): String
~ setUp(): void
+ timeParse(int): void
BagOfPrimitives
+ hashCode(): int
+ getExpectedJson(): String
+ equals(Object): boolean
+ getIntValue(): int
+ toString(): String
1
bag
1
CollectionsDeserializationBenchmark
~ setUp(): void
+ timeCollectionsReflectionStreaming(int): void
+ timeCollectionsDefault(int): void
+ timeCollectionsStreaming(int): void
+ main(String[]): void
SerializationBenchmark
~ setUp(): void
+ timeObjectSerialization(int): void
+ main(String[]): void
BagOfPrimitivesDeserializationBenchmark
~ setUp(): void
+ timeBagOfPrimitivesReflectionStreaming(int): void
+ timeBagOfPrimitivesDefault(int): void
+ timeBagOfPrimitivesStreaming(int): void
+ main(String[]): void

PROTO

https://app.diagrams.net/?&highlight=0000ff&edit=_blank&layers=1&nav=1&title=exported_from_idea.drawio#G1eRQvUuo1yKYx86_ooAi5gzyFYyxIPa0i#%7B%2

2pageId%22%3A%22mXHqL8jpDIJoaNkz4i3R%22%7D


Annotations
+ registerAllExtensions(ExtensionRegistryLite): void
+ registerAllExtensions(ExtensionRegistry): void
+ getDescriptor(): FileDescriptor
Bag
+ registerAllExtensions(ExtensionRegistry): void
+ getDescriptor(): FileDescriptor
+ registerAllExtensions(ExtensionRegistryLite): void
ProtosWithAnnotationsTest
+ setUp(): void
+ testProtoWithAnnotations_serialize(): void
+ testProtoWithAnnotations_deserializeUnrecognizedEnumNumber(): void
+ testProtoWithAnnotations_deserialize(): void
+ testProtoWithAnnotations_deserializeUnrecognizedEnumValue(): void
+ testProtoWithAnnotations_deserializeWithEnumNumbers(): void
+ testProtoWithAnnotations_deserializeUnknownEnumValue(): void
ProtosWithComplexAndRepeatedFieldsTest
+ testSerializeRepeatedFields(): void
+ setUp(): void
+ testDeserializeRepeatedFieldsProto(): void
+ testSerializeDifferentCaseFormat(): void
+ testDeserializeDifferentCaseFormat(): void
ProtoTypeAdapter
+ serialize(Message, Type, JsonSerializationContext): JsonElement
- findValueByNameAndExtension(EnumDescriptor, JsonElement): EnumValueDescriptor
+ newBuilder(): Builder
- getEnumValue(EnumValueDescriptor): Object
- getCustSerializedEnumValue(EnumValueOptions, String): String
+ deserialize(JsonElement, Type, JsonDeserializationContext): Message
- getCustSerializedName(FieldDescriptor): String
- getCachedMethod(Class<?>, String, Class<?>[]): Method
ProtosWithPrimitiveTypesTest
+ testSerializeProto(): void
+ testDeserializeWithExplicitNullValue(): void
+ testDeserializeProto(): void
+ testDeserializeEmptyProto(): void
+ testSerializeEmptyProto(): void
+ setUp(): void
ProtosWithAnnotationsAndJsonNamesTest
- roundTrip(Gson, Gson, String): String
- roundTrip(Gson, Gson, ProtoWithAnnotationsAndJsonNames): ProtoWithAnnotationsAndJsonNames

Work Distribution

M. Ahmad – Codebase setup, test case verification

Hassan Tariq – Documentation, design defect analysis

Umair Azeem – Benchmark enhancements, test restructuring

Ahmad Khakan – Adapter modularization, debugging support