GSON_JSON_deserialization.

Page 5 has GSON download and example code.

http://www.javacreed.com/gson-deserialiser-example/

A Simple Example

Let's say we have the following JSON object, where it contains a popular Java book (Amazon) title written by two, well known, authors.

```
{
  'title':    'Java Puzzlers: Traps, Pitfalls, and Corner Cases',
  'isbn-10':  '032133678X',
  'isbn-13':  '978-0321336781',
  'authors':  ['Joshua Bloch', 'Neal Gafter']
}
```

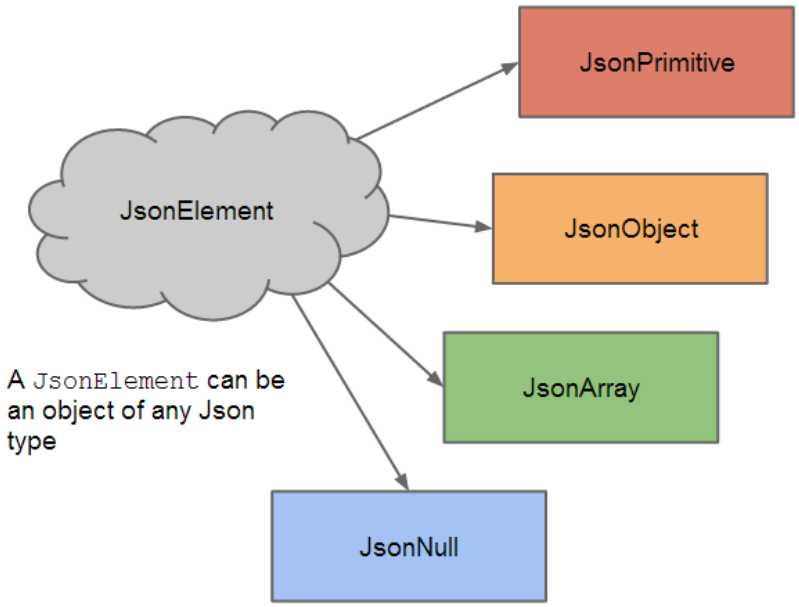
The above JSON comprise four fields, one of which is an array. These fields represent our book. Using the methods discussed in the Simple Gson Example article would create a problem. By default, Gson expects to find variable names in Java with the same name as that found in JSON. Therefore, we should have a class with the following field names: `title`, `isbn-10`, `isbn-13` and `authors`. But names in Java cannot contain the minus sign (`-`) as described in the Java Language Specification (Chapter 6).

Using the `JsonDeserializer`, we have full control over how JSON is parsed as we will see in the following example. Alternatively we can use annotations as described in Gson Annotations Example article. Annotations provide less control, but are simpler to use and understand. With that said, annotations have their limitations too and cannot address all problems described here.

Consider the following simple Java object.

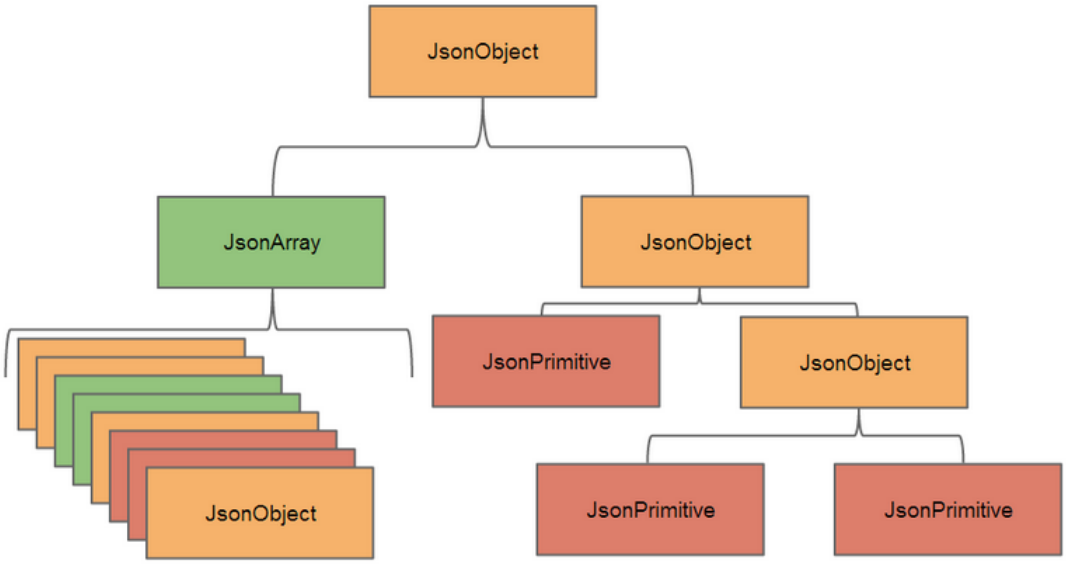
```
package com.javacreed.examples.gson.part1;

public class Book {

  private String[] authors;
  private String isbn10;
  private String isbn13;
  private String title;

  public String[] getAuthors() {
    return authors;
  }

  public String getIsbn10() {
    return isbn10;
  }

  public String getIsbn13() {
    return isbn13;
  }

  public String getTitle() {
    return title;
  }

  public void setAuthors(final String[] authors) {
    this.authors = authors;
  }

  public void setIsbn10(final String isbn10) {
    this.isbn10 = isbn10;
  }

  public void setIsbn13(final String isbn13) {
    this.isbn13 = isbn13;
  }

  public void setTitle(final String title) {
    this.title = title;
  }

  @Override
  public String toString() {
    final StringBuilder formatted = new StringBuilder();
```

```
    formatted.append(title);
    if (isbn10 != null) {
      formatted.append(" [ISBN-10: ").append(isbn10).append("]");
    }
    if (isbn13 != null) {
      formatted.append(" [ISBN 13: ").append(isbn13).append("]");
    }
    formatted.append("\nWritten by:");
    for (final String author : authors) {
      formatted.append("\n  >> ").append(author);
    }

    return formatted.toString();
  }
}
```



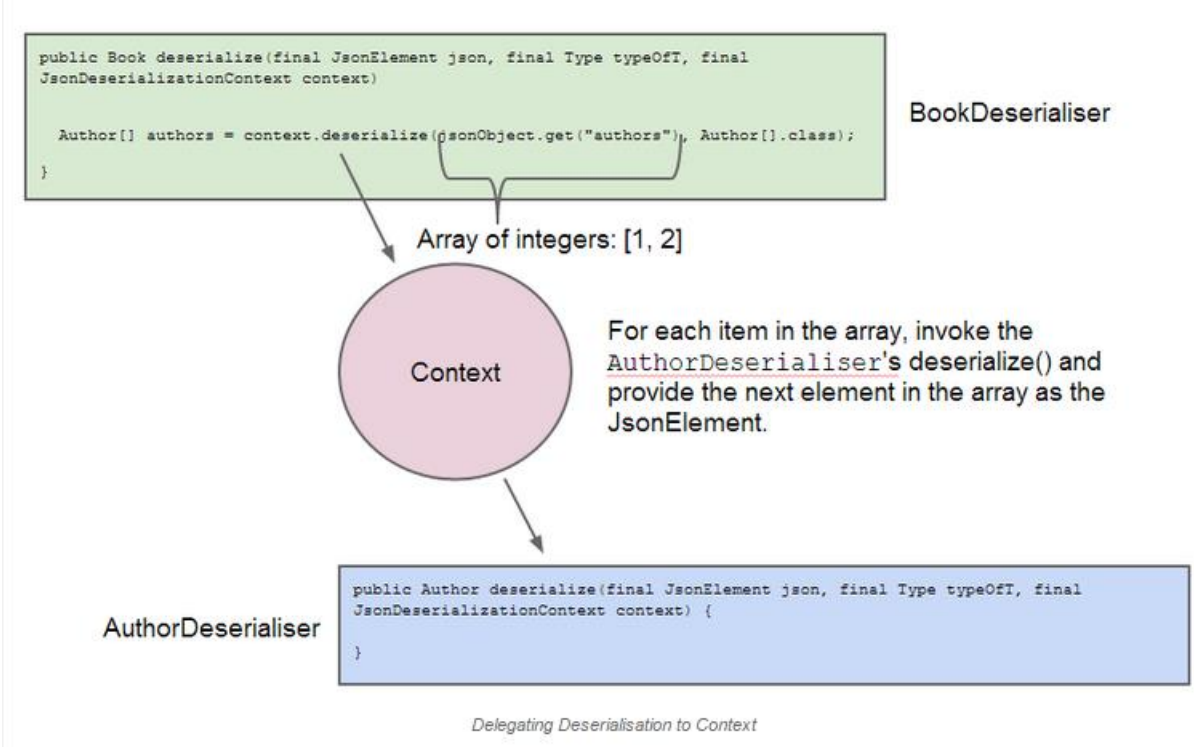
Types of JsonElement

```
package com.javacreed.examples.gson.part2;

import java.lang.reflect.Type;

import com.google.gson.JsonDeserializationContext;
import com.google.gson.JsonDeserializer;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParseException;

public class BookDeserializer implements JsonDeserializer<Book> {

  @Override
  public Book deserialize(final JsonElement json, final Type typeOfT, final JsonDeserializationContext context)
      throws JsonParseException {
    final JsonObject jsonObject = json.getAsJsonObject();

    // Delegate the deserialisation to the AuthorDeserializer class
    final Author[] authors = context.deserialize(jsonObject.get("authors"), Author[].class);

    final Book book = new Book();
    book.setTitle(jsonObject.get("title").getAsString());
    book.setIsbn(jsonObject.get("isbn").getAsString());
    book.setAuthors(authors);
    return book;
  }
}
```



Json Object Hierarchy

```
package com.javacreed.examples.gson.part2;

import java.lang.reflect.Type;

import com.google.gson.JsonDeserializationContext;
import com.google.gson.JsonDeserializer;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParseException;

public class AuthorDeserializer implements JsonDeserializer<Author> {

  @Override
  public Author deserialize(final JsonElement json, final Type typeOfT, final JsonDeserializationContext
context)
      throws JsonParseException {
    final JsonObject jsonObject = json.getAsJsonObject();

    final Author author = new Author();
    author.setId(jsonObject.get("id").getAsInt());
    author.setName(jsonObject.get("name").getAsString());
    return author;
  }
}
```



Delegating Deserialisation to Context

```
public class Author {

  private int id;
  private String name;

  public int getId() {
    return id;
  }

  public String getName() {
    return name;
  }

  public void setId(final int id) {
    this.id = id;
  }

  public void setName(final String name) {
    this.name = name;
  }

  @Override
  public String toString() {
    return String.format("[%d] %s", id, name);
  }
}
```

Maven

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.3.1</version>
</dependency>
```

GIT

```
git clone https://github.com/google/gson.git
```

Grails

https://grails.org/plugin/gson

JAR

http://grepcode.com/snapshot/repo1.maven.org/maven2/com.google.code.gson/gson/2.3.1

GSON API

https://google-gson.googlecode.com/svn/trunk/gson/docs/javadocs/com/google/gson/Gson.html

```
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonObject;
import com.google.gson.reflect.TypeToken;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Type;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;

// https://sites.google.com/site/gson/gson-user-guide
// https://sites.google.com/site/gson/streaming
// http://www.mkyong.com/java/gson-streaming-to-read-and-write-json/
/*
 * SVN information
 * $Revision: 3773 $
 * $Author: ftrujillo $
 * $Date: 2015-09-21 13:03:46 -0600 (Mon, 21 Sep 2015) $
 * $HeadURL:
http://svn/NSG/comp_ssd/software/trunk/NetBeansProjects/Examples/CreateWebApp/src/main/java/org/fjt/support/Json
Utils.java $
 *
 */
public class JsonUtils {
    private static final long serialVersionUID = 1436553694L;


    public JsonUtils() {
    }

    /**
     * This will serialize ANY object.
     *
     * @param <T>      Any object type
     * @param obj      object reference
     * @return Object
     */
    public static <T> String toJson(T obj) {
        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.setPrettyPrinting();
        gsonBuilder.serializeNulls();
        Gson gson = gsonBuilder.create();
        String json = gson.toJson(obj);
```

```
        return (json);
    }

    // fromJson is harder to create a generic method for due to Class<T> mapping.
    // see each class for fromJson() methods.
    public static String readJsonFromFile(String fileName) throws JsonUtilsException {
        String jsonStr = null;
        try {
            // read JSON file as String
            File file = new File(fileName);

            if (file.exists() == false) {
                String msg = "File does NOT exist! => " + fileName;
                throw new JsonUtilsException(msg);
            }

            System.err.println("Reading JSON file => " + file.getCanonicalPath() + "\n");
            jsonStr = new String(Files.readAllBytes(Paths.get(fileName)));

            //String compactJsonStr = JsonUtils.toCompactFormat(jsonStr);
        } catch (IOException ex) {
            String msg = "IOExeption from fileName => " + fileName + " for class => " +
JsonUtils.class.getName();
            msg += ex.getMessage();
            throw new JsonUtilsException(msg);
        }
        return (jsonStr);
    }

    /**
     * Write Json or any String to a file.
     *
     * @param jsonStr               Json String input
     * @param fileName              File Name output
     * @throws JsonUtilsException   If it can not find file.
     */
    public static void writeJsonToFile(String jsonStr, String fileName) throws JsonUtilsException {
        // write JSON String to file
        File file = new File(fileName);
        try {
            if (!file.exists()) {
                file.createNewFile();
            } else {
                file.delete();
                file.createNewFile();
            }

            try (BufferedWriter bw = new BufferedWriter(new FileWriter(file.getAbsoluteFile()))) {
                bw.write(jsonStr);
            }
            System.err.println("Wrote JSON to file => " + file.getCanonicalPath() + "\n");
        } catch (IOException ex) {
            String msg = "IOExeption from fileName => " + fileName + " for class => " +
JsonUtils.class.getName();
            msg += ex.getMessage();
            throw new JsonUtilsException(msg);
        }
    }

    public static String toPrettyFormat(String jsonString) {
        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.setPrettyPrinting();
        gsonBuilder.serializeNulls();
        Gson gson = gsonBuilder.create();
        String prettyJson;

        if (jsonString.trim().startsWith("{")) {
            // java.lang.reflect.Type
            Type type = new TypeToken<JsonObject>() {
            }.getType();
            JsonObject jsonObject = gson.fromJson(jsonString, type);
            prettyJson = gson.toJson(jsonObject);
        } else {
            // java.lang.reflect.Type
            Type type = new TypeToken<ArrayList<JsonObject>>() {
```

```
            }.getType();
            ArrayList<JsonObject> jsonListObjects = gson.fromJson(jsonString, type);
            prettyJson = gson.toJson(jsonListObjects);
        }

        return prettyJson;
    }

    public static String toCompactFormat(String jsonString) {
        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.serializeNulls();
        Gson gson = gsonBuilder.create();
        String compactJson;

        if (jsonString.trim().startsWith("{")) {
            // java.lang.reflect.Type
            Type type = new TypeToken<JsonObject>() {
            }.getType();
            JsonObject jsonObject = gson.fromJson(jsonString, type);
            compactJson = gson.toJson(jsonObject);
        } else {
            // java.lang.reflect.Type
            Type type = new TypeToken<ArrayList<JsonObject>>() {
            }.getType();
            ArrayList<JsonObject> jsonListObjects = gson.fromJson(jsonString, type);
            compactJson = gson.toJson(jsonListObjects);
        }

        return compactJson;
    }

}
```