REST client connectivity:

URL url = new URL(urlString);

URLConnection conn = url.openConnection();

conn.setDoOutput(true);

conn.setRequestProperty ("Authorization", encodedCredentials);

OutputStreamWriter writer = new OutputStreamWriter(conn.getOutputStream());

writer.write(data);

writer.flush();

String line;

BufferedReader reader = new BufferedReader(new

                                 InputStreamReader(conn.getInputStream()));

while ((line = reader.readLine()) != null) {

  System.out.println(line);

}

writer.close();

reader.close();

**Json String parser**

* **Model should serializable**
* class Root {  
   @SerializedName("Outer")  
   public Map<Integer, Outer> outer;  
   @Override  
   public String toString() {  
   return "Root[outer=" + this.outer + "]";  
   }  
  }
* class Outer {  
   @SerializedName("Attr1")  
   public int attr1;  
   @SerializedName("Attr2")  
   public int attr2;  
   @Override  
   public String toString() {  
   return "Outer[attr1=" + this.attr1 + ", attr2=" + this.attr2 + "]";  
   }  
  }

**How to Parse JSON Array with Gson**

You can parse the JSONArray directly, don't need to wrap your Post class with PostEntity one more time and don't need new JSONObject().toString() either:

Gson gson = new Gson();  
String jsonOutput = "Your JSON String";  
Type listType = new TypeToken<List<Post>>(){}.getType();  
List<Post> posts = gson.fromJson(jsonOutput, listType);

Hope that helps.

**Java Gson parse Json object to array**

***TL;DR:****See "Using Deserializer" section at the bottom for parsing straight to array.*

That JSON does not contain any arrays. An array would use the [...] JSON syntax.

Normally, a JSON object would map to a POJO, with the name in the name/value pairs mapping to a field of the POJO.

However, a JSON object can also be mapped to a Map, which is especially useful when the names are dynamic, since POJO fields are static.

**Using Map**

The JSON object with numeric values as names can be mapped to a Map<Integer, ?>, e.g. to parse that JSON to POJOs, do it like this:

class Root {  
 @SerializedName("Outer")  
 public Map<Integer, Outer> outer;  
 @Override  
 public String toString() {  
 return "Root[outer=" + this.outer + "]";  
 }  
}

class Outer {  
 @SerializedName("Attr1")  
 public int attr1;  
 @SerializedName("Attr2")  
 public int attr2;  
 @Override  
 public String toString() {  
 return "Outer[attr1=" + this.attr1 + ", attr2=" + this.attr2 + "]";  
 }  
}

*Test*

Gson gson = new GsonBuilder().create();  
Root root;  
try (BufferedReader in = Files.newBufferedReader(Paths.get("test.json"))) {  
 root = gson.fromJson(in, Root.class);  
}  
System.out.println(root);

*Output*

Root[outer={0=Outer[attr1=12345, attr2=67890], 1=Outer[attr1=54321, attr2=9876]}]

**Get as Array**

You can then add a helper method to the Root class to get that as an array:

public Outer[] getOuterAsArray() {  
 if (this.outer == null)  
 return null;  
 if (this.outer.isEmpty())  
 return new Outer[0];  
 int maxKey = this.outer.keySet().stream().mapToInt(Integer::intValue).max().getAsInt();  
 Outer[] arr = new Outer[maxKey + 1];  
 this.outer.forEach((k, v) -> arr[k] = v);  
 return arr;  
}

*Test*

System.out.println(Arrays.toString(root.getOuterAsArray()));

*Output*

[Outer[attr1=12345, attr2=67890], Outer[attr1=54321, attr2=9876]]

**Using Deserializer**

However, it would likely be more useful if the conversion to array is done while parsing, so you need to write a JsonDeserializer and tell Gson about it using @JsonAdapter:

class Root {  
 @SerializedName("Outer")  
 @JsonAdapter(OuterArrayDeserializer.class)  
 public Outer[] outer;  
  
 @Override  
 public String toString() {  
 return "Root[outer=" + Arrays.toString(this.outer) + "]";  
 }  
}

class OuterArrayDeserializer implements JsonDeserializer<Outer[]> {  
 @Override  
 public Outer[] deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext context) throws JsonParseException {  
 // Parse JSON array normally  
 if (json.isJsonArray())  
 return context.deserialize(json, Outer[].class);  
  
 // Parse JSON object using names as array indexes  
 JsonObject obj = json.getAsJsonObject();  
 if (obj.size() == 0)  
 return new Outer[0];  
 int maxKey = obj.keySet().stream().mapToInt(Integer::parseInt).max().getAsInt();  
 Outer[] arr = new Outer[maxKey + 1];  
 for (Entry<String, JsonElement> e : obj.entrySet())  
 arr[Integer.parseInt(e.getKey())] = context.deserialize(e.getValue(), Outer.class);  
 return arr;  
 }  
}

Same Outer class and test code as above.

*Output*

Root[outer=[Outer[attr1=12345, attr2=67890], Outer[attr1=54321, attr2=9876]]]

**Using GSON to parse a JSON array and Object**

I think ResultOsrm should hold list of Waypoint and class Waypoint will hold the data

public class ResultOsrm  
{  
 public List<Waypoint> waypoints;  
}  
  
public class Waypoint  
{  
 public int waypoint\_index;  
 public int trips\_index;  
 public String hint;  
 public String name;  
 public List<float> location;  
}

waypoint\_index is a variable in Waypoint, not a list by itself.

**How to parse Json array of different objects with Gson?**

I think there are lots of simmilar questions on SO. One, Two

One way to parse this is to use simple

Object[] result = new Gson().fromJson(json, Object[].class);

But this will give you objects of LinkedTreeMap<Integer, LinkedTreeMap<String, String>> or something like this. You can use it, but its kinda hard and you will also have problems with your integers comming as doubles.

The other approach is to create custom interface or abstract class with TypeName field if you need it:

private interface CheckInterface{}

and implement it with every POJO classes of object types you have:

private static class CheckEveryDayBase implements CheckInterface{  
 private String StartDate;  
 private String EndDate;  
 private int Interval;  
 private int HolidayCondition;  
}  
  
private static class CheckSpecificDday implements CheckInterface{  
 private String SpecificDay;  
 private int Lunar;  
}  
  
private static class CheckEveryDayDday extends CheckEveryDayBase{  
 private String StartOption;  
}  
  
private static class CheckEveryDdayOfWeek extends CheckEveryDayBase{  
 private String SpecificDayOfWeek;  
}  
  
private static class CheckEveryMonthSpecificDday extends CheckEveryDayBase{  
 private String SpecificDD;  
}  
  
private static class CheckEveryYearWeek extends CheckEveryDayBase{  
 private String SpecificMMnthWeek;  
}

Then create custom desrializer for your CheckInterface:

public static class CheckInterfaceDeserializer implements JsonDeserializer<CheckInterface>{  
  
 @Override  
 public CheckInterface deserialize(JsonElement json, Type typeOfT,  
 JsonDeserializationContext context) throws JsonParseException {  
 JsonObject jObject = (JsonObject) json;  
 JsonElement typeObj = jObject.get("TypeName");  
  
 if(typeObj!= null ){  
 String typeVal = typeObj.getAsString();  
  
 switch (typeVal){  
 case "CheckSpecificDday":  
 return context.deserialize(json, CheckSpecificDday.class);  
 case "CheckEveryDayDday":  
 return context.deserialize(json, CheckEveryDayDday.class);  
 case "CheckEveryDdayOfWeek":  
 return context.deserialize(json, CheckEveryDdayOfWeek.class);  
 case "CheckEveryMonthSpecificDday":  
 return context.deserialize(json, CheckEveryMonthSpecificDday.class);  
 case "CheckEveryYearWeek":  
 return context.deserialize(json, CheckEveryYearWeek.class);  
 }  
 }  
  
 return null;  
 }  
}

Here is how you can use this:

GsonBuilder builder = new GsonBuilder();  
  
// Register custom deserializer for CheckInterface.class  
builder.registerTypeAdapter(CheckInterface.class, new CheckInterfaceDeserializer());  
Gson gson = builder.create();  
  
CheckInterface[] result2 = gson.fromJson(json, CheckInterface[].class);

**Using Gson in Kotlin to parse JSON array**

You need to change parameter in your fromJson() function call like following:

val weatherList: List<WeatherObject> = gson.fromJson(stringReader , Array<WeatherObject>::class.java).toList()

You need to pass Array<WeatherObject>::class.java for class type and then convert result into List. No need to change registerTypeAdapter() function call.

Check following code:

fun getWeatherObjectFromJson(jsonStr: String): List<WeatherObject> {  
  
 var stringReader: StringReader = StringReader(jsonStr)  
 var jsonReader: JsonReader = JsonReader(stringReader)  
  
 val gsonBuilder = GsonBuilder().serializeNulls()  
 gsonBuilder.registerTypeAdapter(WeatherObject::class.java, WeatherDeserializer())  
 val gson = gsonBuilder.create()  
  
 val weatherList: List<WeatherObject> = gson.fromJson(stringReader , Array<WeatherObject>::class.java).toList()  
  
 return weatherList  
 }

**Gson is unable to parse a json array string located in an json object AS a JsonArray**

I understand the REST API response is bad and is violating the JSON syntax.  
Solution is to correct the REST API, but in my scenario unfotunately i cannot request for API correction so i wrote a util at my end to clean the jsonString.

Posting it here if it helps anyone :

/\*\*  
 \* @param malformedArrayKey  
 \* - Name of the key in the JSON object that has a malformed array  
 \* for e.g consider following JSON object having a bad formed array  
 \* <pre>  
 \* {  
 \* "task": "findRecords",  
 \* "foundRecords": "[1234567, 11234512]",  
 \* }  
 \* </pre>  
 \* @param jsonString  
 \* - String representation of the JSON object containing the malformed array  
 \* @return - json string having well formed array against the key {@code malformedArrayKey} supplied  
 \* <pre>  
 \* {  
 \* "task": "findRecords",  
 \* "foundRecords": [1234567, 11234512]  
 \* }  
 \* </pre>  
 \*/  
 public static String formatMalformedArray(String malformedArrayKey, String jsonString) {  
 JsonObject jsonObj = gson.fromJson(jsonString, JsonObject.class);  
 // get the faulty key value  
 String malformedArrayKeyValue = jsonObj.get(malformedArrayKey)  
 .getAsString();  
 // drop it  
 jsonObj.remove(malformedArrayKey);  
 // create a array out of the malformed array string  
 JsonArray jsonArray = gson.fromJson(malformedArrayKeyValue, JsonArray.class);  
 // add the array back to the object  
 jsonObj.add(malformedArrayKey, jsonArray);  
 // now convert it into a well formed json string  
 return jsonObj.toString();  
 }

The method is quite basic but it satisifes my use case.

You can use the standard Java SE APIs:

private void updateCustomer(Customer customer) {

try {

URL url = new URL("http://www.example.com/customers");

HttpURLConnection connection = (HttpURLConnection) url.openConnection();

connection.setDoOutput(true);

connection.setInstanceFollowRedirects(false);

connection.setRequestMethod("PUT");

connection.setRequestProperty("Content-Type", "application/xml");

OutputStream os = connection.getOutputStream();

jaxbContext.createMarshaller().marshal(customer, os);

os.flush();

connection.getResponseCode();

connection.disconnect();

} catch(Exception e) {

throw new RuntimeException(e);

}

}