**1. JSON is a document format used to encode information that is both human-readable and**

**machine-readable. JSON format is explained at http://json.org. Please write a JSON**

**parser that accepts an input JSON string and produces a Map output structure. Do not**

**use any existing library to do the parsing.**

**Example input JSON:**

**{**

**"debug" : "on",**

**"window" : {**

**"title" : "sample",**

**"size": 500**

**}**

**}**

**The parser may be a static function:**

**public class JSONParser {**

**public static Map<String,Object> parse(String json) {**

**...**

**}**

**}**

**The corresponding output for the input JSON should be:**

**Map<string, Object> output = JsonParser.parse(input);**

**assert output.get("debug").equals("on");**

**assert (Map<string, Object>(output.get("window")).get("title").equals("sample")**

**assert (Map<String, Object>(output.get("window")).get("size").equals(500)**

**Solution:**

**package** defaultpackage;

**import** java.util.\*;

**public** **class** JSONParser {

**public** **static** Map<String, Object> parse(String json) {

*currentIndex* = 0;

**return** (Map<String, Object>) *parseValue*(json);

}

**private** **static** **int** *currentIndex*;

**private** **static** Object parseValue(String json) {

**char** currentChar = json.charAt(*currentIndex*);

**if** (currentChar == '{') {

**return** *parseObject*(json);

} **else** **if** (currentChar == '[') {

**return** *parseArray*(json);

} **else** **if** (currentChar == '"') {

**return** *parseString*(json);

} **else** **if** (currentChar == 't' || currentChar == 'f') {

**return** *parseBoolean*(json);

} **else** **if** (currentChar == 'n') {

**return** *parseNull*(json);

} **else** **if** (currentChar == '-' || Character.*isDigit*(currentChar)) {

**return** *parseNumber*(json);

} **else** {

**throw** **new** RuntimeException("Invalid JSON format");

}

}

**private** **static** Map<String, Object> parseObject(String json) {

Map<String, Object> result = **new** HashMap<>();

*currentIndex*++; // Skip the opening brace

**while** (*currentIndex* < json.length()) {

*skipWhitespace*(json);

**if** (json.charAt(*currentIndex*) == '}') {

*currentIndex*++; // Skip the closing brace

**return** result;

} **else** **if** (json.charAt(*currentIndex*) == '"') {

String key = *parseString*(json);

*skipWhitespace*(json);

**if** (json.charAt(*currentIndex*) == ':') {

*currentIndex*++; // Skip the colon

*skipWhitespace*(json);

Object value = *parseValue*(json);

result.put(key, value);

*skipWhitespace*(json);

**if** (json.charAt(*currentIndex*) == ',') {

*currentIndex*++; // Skip the comma

}

} **else** {

**throw** **new** RuntimeException("Invalid JSON format");

}

} **else** {

**throw** **new** RuntimeException("Invalid JSON format");

}

}

**throw** **new** RuntimeException("Invalid JSON format");

}

**private** **static** List<Object> parseArray(String json) {

List<Object> result = **new** ArrayList<>();

*currentIndex*++; // Skip the opening bracket

**while** (*currentIndex* < json.length()) {

*skipWhitespace*(json);

**if** (json.charAt(*currentIndex*) == ']') {

*currentIndex*++; // Skip the closing bracket

**return** result;

} **else** {

Object value = *parseValue*(json);

result.add(value);

*skipWhitespace*(json);

**if** (json.charAt(*currentIndex*) == ',') {

*currentIndex*++; // Skip the comma

}

}

}

**throw** **new** RuntimeException("Invalid JSON format");

}

**private** **static** String parseString(String json) {

StringBuilder value = **new** StringBuilder();

*currentIndex*++; // Skip the opening double quote

**while** (*currentIndex* < json.length()) {

**char** currentChar = json.charAt(*currentIndex*);

**if** (currentChar == '"') {

*currentIndex*++; // Skip the closing double quote

**return** value.toString();

} **else** **if** (currentChar == '\\') {

*currentIndex*++; // Skip the backslash

**if** (*currentIndex* < json.length()) {

**char** escapedChar = json.charAt(*currentIndex*);

**if** (escapedChar == '"' || escapedChar == '\\' || escapedChar == '/' || escapedChar == 'b'

|| escapedChar == 'f' || escapedChar == 'n' || escapedChar == 'r' || escapedChar == 't') {

value.append(escapedChar);

*currentIndex*++; // Skip the escaped character

} **else** **if** (escapedChar == 'u') {

**if** (*currentIndex* + 4 < json.length()) {

String unicode = json.substring(*currentIndex* + 1, *currentIndex* + 5);

**try** {

**int** codePoint = Integer.*parseInt*(unicode, 16);

value.append((**char**) codePoint);

*currentIndex* += 5; // Skip the escaped unicode sequence

} **catch** (NumberFormatException e) {

**throw** **new** RuntimeException("Invalid escaped unicode sequence");

}

} **else** {

**throw** **new** RuntimeException("Invalid escaped unicode sequence");

}

} **else** {

**throw** **new** RuntimeException("Invalid escaped character");

}

}

} **else** {

value.append(currentChar);

*currentIndex*++;

}

}

**throw** **new** RuntimeException("Invalid JSON format");

}

**private** **static** **boolean** parseBoolean(String json) {

**if** (json.startsWith("true", *currentIndex*)) {

*currentIndex* += 4; // Skip "true"

**return** **true**;

} **else** **if** (json.startsWith("false", *currentIndex*)) {

*currentIndex* += 5; // Skip "false"

**return** **false**;

} **else** {

**throw** **new** RuntimeException("Invalid JSON format");

}

}

**private** **static** Object parseNull(String json) {

**if** (json.startsWith("null", *currentIndex*)) {

*currentIndex* += 4; // Skip "null"

**return** **null**;

} **else** {

**throw** **new** RuntimeException("Invalid JSON format");

}

}

**private** **static** Number parseNumber(String json) {

**int** startIndex = *currentIndex*;

**while** (*currentIndex* < json.length()) {

**char** currentChar = json.charAt(*currentIndex*);

**if** (!Character.*isDigit*(currentChar) && currentChar != '-' && currentChar != '.' && currentChar != 'e' && currentChar != 'E') {

String numberStr = json.substring(startIndex, *currentIndex*);

**if** (numberStr.contains(".") || numberStr.toLowerCase().contains("e")) {

**return** Double.*parseDouble*(numberStr);

} **else** {

**return** Long.*parseLong*(numberStr);

}

}

*currentIndex*++;

}

String numberStr = json.substring(startIndex, *currentIndex*);

**if** (numberStr.contains(".") || numberStr.toLowerCase().contains("e")) {

**return** Double.*parseDouble*(numberStr);

} **else** {

**return** Long.*parseLong*(numberStr);

}

}

**private** **static** **void** skipWhitespace(String json) {

**while** (*currentIndex* < json.length() && Character.*isWhitespace*(json.charAt(*currentIndex*))) {

*currentIndex*++;

}

}

**public** **static** **void** main(String[] args) {

String jsonString = "{\n" +

" \"area-items\": {\n" +

" \"2\": [\n" +

" {\n" +

" \"id\": -250384452623414200,\n" +

" \"title\": \"Probe Agency NIA Convicts 5 Members Of Banned Outfit In Bijnor Blast Case\",\n" +

" \"url\": \"https://www.ndtv.com/india-news/nia-court-convicts-5-simi-members-in-bijnor-ied-blast-case-3120426\",\n" +

" \"description\": \"A special National Investigation Agency (NIA) court in Lucknow has convicted five members of the banned organisation SIMI to commit terrorist acts and sentenced them to rigorous imprisonment in the 2014 Bijnor blast case\",\n" +

" \"publishEpochMillis\": 1656742080000,\n" +

" \"expirationEpochMillis\": 1656914880000,\n" +

" \"modifiedEpochMillis\": null,\n" +

" \"tags\": [\n" +

" \"simi\",\n" +

" \"national investigation agency\",\n" +

" \"bijnor blast case\"\n" +

" ],\n" +

" \"categories\": [\n" +

" \"news\"\n" +

" ],\n" +

" \"flags\": [],\n" +

" \"blocked\": false,\n" +

" \"thumbnail-url\": \"https://c.ndtvimg.com/2022-06/8jh8j9f8\_police-generic-\_625x300\_01\_June\_22.jpg\",\n" +

" \"publish-date\": \"2022-07-02 11:38:00\",\n" +

" \"expiration-date\": \"2022-07-04 11:38:00\",\n" +

" \"modified-date\": null,\n" +

" \"flag-update-time\": null\n" +

" }\n" +

" ]\n" +

" }\n" +

"}";

Object result = *parse*(jsonString);

String output = *formatOutput*(result);

System.***out***.println(output);

}

**private** **static** String formatOutput(Object obj) {

**if** (obj == **null**) {

**return** "null";

} **else** **if** (obj **instanceof** Map) {

StringBuilder builder = **new** StringBuilder("{");

Map<?, ?> map = (Map<?, ?>) obj;

**for** (Map.Entry<?, ?> entry : map.entrySet()) {

builder.append("\"").append(entry.getKey()).append("\":");

builder.append(*formatOutput*(entry.getValue())).append(",");

}

**if** (!map.isEmpty()) {

builder.setLength(builder.length() - 1); // Remove the last comma

}

builder.append("}");

**return** builder.toString();

} **else** **if** (obj **instanceof** List) {

StringBuilder builder = **new** StringBuilder("[");

List<?> list = (List<?>) obj;

**for** (Object item : list) {

builder.append(*formatOutput*(item)).append(",");

}

**if** (!list.isEmpty()) {

builder.setLength(builder.length() - 1); // Remove the last comma

}

builder.append("]");

**return** builder.toString();

} **else** **if** (obj **instanceof** String) {

**return** "\"" + obj + "\"";

} **else** {

**return** obj.toString();

}

}

}