Exercises 10

Task 10.1. Deserialization Vulnerability

We discussed the problem of deserialization vulnerabilities in the lecture, but it would be good to understand at least one practical case to see how this type of security hole can be exploited.

Read the article "Deserialization of untrusted data". Pick up any reference mentioned at the bottom of the page and study the case described there. Provide a concise description of the case, including:

- a short case name;
- the description of vulnerability;
- what exactly an attacker can do;
- how technically this attack can be performed and how exactly serialization is exploited.

Task 10.2. Use Datamuse

Use Datamuse REST API to extract and print words that are spelled like the word provided by the user. Start with the code provided in ex10_rest.java and modify it as follows:

- You will need to use `sp` function of the Datamuse API.
- Since user input in this case may include special characters like `?` or spaces, implement a proper encoding of query string parameters using `URLEncoder` class as explained here.
- Use Gson library to convert a JSON response into Java objects. Note that you can get a precompiled Gson JAR file from the project's Maven repository (click **Downloads** link). Read also this tutorial on deserialization of arrays of objects.

Example input data: `sal?on`

Example output data:

```
salmon
saloon
salton
```

Task 10.3. RMI Chat

Rewrite the chat program (Task 9.4) using RMI. It would be an easier method comparing to using sockets.

Task 10.4. RMI Word Guessing Game

Rewrite the word guessing game (Task 9.3) using RMI. This time, let's impose no limitations on the number of concurrent clients.

Hint. Probably, the simplest remote object for this purpose has to implement just two public methods:

```
// creates a new game for the client and returns a unique game ID
int InitGame();

// returns the current string to the client playing the given game <gameId>
// as a response to the client's chosen next character <nextChar>
String ShowWord(char nextChar, int gameId);
```

For proper concurrency, you need to keep separate game states (the chosen word and the list of guessed characters) for each game ID. A `TreeMap` structure would be a simple choice for storing all game sessions in computer memory. Once a certain game is over, remove its state.