Ex10

s1260119 Yuta Tomiyama

Task 10.1
The Forgotten Bug Class
Deserialize method that have side-effect may down system
Attacker can invoke any serializable class deserialize code
If serializable class readObject method have side-effect code for example read file, OutOfMemoryException may occur when deserialize too many.

Task 10.2
```java
import com.google.gson.Gson;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

class ex10_rest {

    private static Gson gson = new Gson();

    static private String call_url(String base_url, String function,
Map<String, String> args) throws Exception {
        String[] escaped = args.entrySet()
                .stream()
                .map(e -> e.getKey() + "=" +
URLEncoder.encode(e.getValue(), StandardCharsets.UTF_8))
                .toArray(String[]::new);
        String argslist = String.join("&", escaped);
        URL url = new URL(base_url + "/" + function + "?" + argslist);
        System.out.println("Calling: " + url);
        InputStreamReader is = new InputStreamReader(url.openStream());
        int c;
        String r = "";
        while ((c = is.read()) != -1)
            r += (char) c;
        return r;
    }

    static public void main(String args[]) throws Exception {
        Map<String, String> queries = new HashMap<>();
        queries.put("sp", "sal?on");
        String response = call_url("https://api.datamuse.com", "words",
queries);
        DataJson[] json = gson.fromJson(response, DataJson[].class);
        System.out.println(Arrays.toString(json));
    }
}

class DataJson {
    String word;
    int score;

    @Override
    public String toString() {
        return "{" +
                "word=\"" + word + '"' +
                ",score=" + score +
                '}';
    }
}
```

Task 10.3
server

```
import java.rmi.Naming;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.List;
import java.util.stream.Collectors;

public class BBServer {
    public static void main(String[] args) {
        try {
            Naming.rebind("BBService", new BoardImpl());
            System.out.println("server start");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

interface Board extends Remote {
    void addMessage(String msg) throws RemoteException;
    String showMessages() throws RemoteException;
}

class BoardImpl extends UnicastRemoteObject implements Board {

    private static final SimpleDateFormat sdf = new
SimpleDateFormat("dd.MM.yyyy hh:mm:ss");
    private final List<Message> messages =
Collections.synchronizedList(new ArrayList<>());

    public BoardImpl() throws RemoteException {}

    @Override
    public void addMessage(String msg) throws RemoteException {
        Message message = new Message(msg);
        messages.add(message);
    }

    @Override
    public String showMessages() throws RemoteException {
        return messages.stream()
                .map(m -> sdf.format(m.createdAt) + '\n' + m.body +
'\n')
                .collect(Collectors.joining("\n"));
    }
}

class Message {
    final String body;
    final Date createdAt;

    public Message(String body) {
        this.body = body;
        createdAt = new Date();
    }
}
```

client

```
import java.rmi.Naming;
import java.util.Scanner;

public class BBClient {
    public static void main(String[] args) {
        String mode = args[0];
        try {
            Board board = (Board)
Naming.lookup("//127.0.0.1/BBService");
            if (mode.equals("add")) {
                Scanner sc = new Scanner(System.in);
                String message = sc.nextLine();
                board.addMessage(message);
            } else if (mode.equals("read")) {
                System.out.println(board.showMessages());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Task 10.4

```
import java.rmi.Naming;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.*;

public class WordGuessingGameRMI {
    public static void main(String[] args) {
        try {
            if (args[0].equals("server")) {
                Naming.rebind("GameService", new RMIGameServerImpl());
                System.out.println("server start");
            } else {
                Scanner sc = new Scanner(System.in);
                RMIGameServer server = (RMIGameServer)
Naming.lookup("//127.0.0.1/GameService");
                int id = server.initGame();
                while (true) {
                    System.out.print("client> ");
                    char input = sc.nextLine().charAt(0);
                    String result = server.showWord(input, id);
                    System.out.println("server> " + result);
                    if (!result.contains("*")) {
                        break;
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

interface RMIGameServer extends Remote {
    int initGame() throws RemoteException;
    String showWord(char nextChar, int gameId) throws RemoteException;
}

class RMIGameServerImpl extends UnicastRemoteObject implements
RMIGameServer {

    private static final List<String> words = Arrays.asList("book",
"block", "follow");
    private final Map<Integer, GameSession> sessions = new TreeMap<>();
    private int id = 0;

    public RMIGameServerImpl() throws RemoteException {}

    @Override
    public synchronized int initGame() throws RemoteException {
        String word = words.get(new Random().nextInt(words.size()));
        sessions.put(id, new GameSession(word));
        return id++;
    }

    @Override
    public String showWord(char nextChar, int gameId) throws
RemoteException {
        GameSession session = sessions.get(gameId);
        String result = session.checkChar(nextChar);
        if (!result.contains("*")) {
            sessions.remove(gameId);
        }
        return result;
    }
}

class GameSession {
    private final String word;
    private final List<Character> hitChars = new ArrayList<>();

    public GameSession(String word) {
        this.word = word;
    }
```

```java
    public String checkChar(char input) {
        if (!hitChars.contains(input) &&
word.contains(String.valueOf(input))) {
            hitChars.add(input);
        }
        StringBuilder resultBuilder = new StringBuilder();
        for (char c : word.toCharArray()) {
            if (hitChars.contains(c)) {
                resultBuilder.append(c);
            } else {
                resultBuilder.append('*');
            }
        }
        String result = resultBuilder.toString();
        return result;
    }
}
```