```
package lesson8;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Random;
import java.util.Timer;
import java.util.TimerTask;

public class Test1 {
	private static final int PORT = 12345;
	private static final int WINNING_SCORE = 30;
	private static final int TURN_TIME_SECONDS = 60;

	private ServerSocket serverSocket;
	private List<ClientHandler> clients = new ArrayList<>();
	private Map<Integer, Integer> scores = new HashMap<>();
	private List<String> katakanaWords = Arrays.asList(
			"apple", "banana", "cherry", "grape", "lemon",
			"melon", "orange", "peach", "plum", "berry");

	private Random random = new Random();
	private int currentPlayerIndex = 0;
	private String currentWord;

	private Timer turnTimer;
	private boolean turnAnswered;

	public static void main(String[] args) {
		new Test1().startServer();
	}

	public void startServer() {
		try {
			serverSocket = new ServerSocket(PORT);
```

```
                System.out.println("Katakana Typing Battle Server started on port " +
PORT);

                new Thread(this::acceptClients).start();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        private void acceptClients() {
            while (true) {
                try {
                    Socket socket = serverSocket.accept();
                    ClientHandler client = new ClientHandler(socket, clients.size());
                    clients.add(client);
                    scores.put(client.getId(), 0);
                    client.sendMessage("Welcome! You are Player " + (client.getId() +
1));
                    broadcast("Player " + (client.getId() + 1) + " joined the game. Total
players: " + clients.size());

                    if (clients.size() >= 2) {
                        broadcast("Game is starting now! First to " +
WINNING_SCORE + " points wins!");
                        startGame();
                        break;
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        private void startGame() {
            nextTurn();
        }

        private void nextTurn() {
            currentPlayerIndex = currentPlayerIndex % clients.size();
            currentWord = katakanaWords.get(random.nextInt(katakanaWords.size()));
            turnAnswered = false;

            broadcast("\nIt's Player " + (currentPlayerIndex + 1) + "'s turn!");
            broadcast("Type this Katakana word: " + currentWord);
```

```
                clients.get(currentPlayerIndex).prompt();

                // Start turn timer
                if (turnTimer != null) {
                        turnTimer.cancel();
                }
                turnTimer = new Timer();
                turnTimer.schedule(new TimerTask() {
                        @Override
                        public void run() {
                                if (!turnAnswered) {
                                        broadcast("⏰ Time's up! Player " + (currentPlayerIndex +
1) + " did not answer in time.");
                                        currentPlayerIndex = (currentPlayerIndex + 1) %
clients.size();
                                        nextTurn();
                                }
                        }
                }, TURN_TIME_SECONDS * 1000L);
        }

        private void handleAnswer(int playerIndex, String answer) {
                if (playerIndex != currentPlayerIndex) {
                        clients.get(playerIndex).sendMessage("⛔ It's not your turn!");
                        return;
                }

                // Mark answered to stop timer
                turnAnswered = true;
                if (turnTimer != null) {
                        turnTimer.cancel();
                }

                System.out.println("Received: [" + answer + "] vs Expected: [" + currentWord +
"]");

                if (answer.trim().equals(currentWord.trim())) {
                        int newScore = scores.get(playerIndex) + 10;
                        scores.put(playerIndex, newScore);
                        broadcast("✅ Player " + (playerIndex + 1) + " answered correctly! Score:
" + newScore);

                        if (newScore >= WINNING_SCORE) {
```

```
                                broadcast("🏆 Player " + (playerIndex + 1) + " wins with " +
newScore + " points!");
                                showFinalScores();
                                endGame();
                                return;
                        }
                } else {
                        broadcast("❌ Player " + (playerIndex + 1) + " answered incorrectly.
Correct word was: " + currentWord);
                }

                currentPlayerIndex = (currentPlayerIndex + 1) % clients.size();
                nextTurn();
        }

        private void broadcast(String message) {
                for (ClientHandler client : clients) {
                        client.sendMessage(message);
                }
        }

        private void showFinalScores() {
                broadcast("\n📊 Final Scores:");
                for (int i = 0; i < clients.size(); i++) {
                        broadcast("Player " + (i + 1) + ": " + scores.get(i) + " points");
                }
        }

        private void endGame() {
                broadcast("\n👋 Thank you for playing!");
                try {
                        serverSocket.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                System.exit(0);
        }

        private class ClientHandler implements Runnable {
                private Socket socket;
                private int id;
                private BufferedReader in;
                private PrintWriter out;
```

```
		public ClientHandler(Socket socket, int id) {
			this.socket = socket;
			this.id = id;
			try {
				in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
				out = new PrintWriter(socket.getOutputStream(), true);
			} catch (IOException e) {
				e.printStackTrace();
			}
			new Thread(this).start();
		}

		public int getId() {
			return id;
		}

		public void sendMessage(String message) {
			out.println(message);
		}

		public void prompt() {
			out.print("Type here: ");
			out.flush();
		}

		@Override
		public void run() {
			try {
				String input;
				while ((input = in.readLine()) != null) {
					handleAnswer(id, input.trim());
				}
			} catch (IOException e) {
				System.out.println("Player " + (id + 1) + " disconnected.");
			}
		}
	}
}
```